Robotics: Science and Systems 2025
Los Angeles, CA, USA, June 21–25, 2025

Dynamic Rank Adjustment in Diffusion Policies for Efficient and Flexible Training

Xiatao Sun[1†], Shuo Yang[2], Yinxing Chen[1], Francis Fan[1], Yiyan (Edgar) Liang[2], Daniel Rakita[1†],
[1]Yale University [2]University of Pennsylvania
[†]Corresponding author {xiatao.sun, daniel.rakita}@yale.edu

***Abstract*—Diffusion policies trained via offline behavioral cloning have recently gained traction in robotic motion generation. While effective, these policies typically require a large number of trainable parameters. This model size affords powerful representations but also incurs high computational cost during training. Ideally, it would be beneficial to *dynamically adjust* the trainable portion as needed, balancing representational power with computational efficiency. For example, while overparameterization enables diffusion policies to capture complex robotic behaviors via offline behavioral cloning, the increased computational demand makes online interactive imitation learning impractical due to longer training time. To address this challenge, we present a framework, called DRIFT, that uses the Singular Value Decomposition to enable dynamic rank adjustment during diffusion policy training. We implement and demonstrate the benefits of this framework in DRIFT-DAgger, an imitation learning algorithm that can seamlessly slide between an offline bootstrapping phase and an online interactive phase. We perform extensive experiments to better understand the proposed framework, and demonstrate that DRIFT-DAgger achieves improved sample efficiency and faster training with minimal impact on model performance. The project website is available at: https://apollo-lab-yale.github.io/25-RSS-DRIFT-website/.**

I. Introduction

Diffusion policies have recently emerged as a powerful paradigm for robotic motion generation, demonstrating impressive performance across various manipulation tasks [3]. Their strong performance is attributed to overparameterization, which has been shown to enhance both the generalization and representational capacity of neural networks [5]. However, this advantage comes with a significant drawback, which is the inefficiency of training overparameterized diffusion models [51]. While the machine learning community has made strides in addressing this inefficiency by leveraging the intrinsic low-rank structure of neural networks [25, 26], these approaches primarily focus on fine-tuning pre-trained models [13, 6]. In robotics, however, policies are often trained from scratch, making these fine-tuning methods unsuitable.

To exploit the intrinsic low-rank structure, training from scratch in robotics necessitates a dynamic approach to balancing the trade-off between overparameterization, which provides strong representational power, and the efficiency gained from reduced-rank training. Ideally, at the beginning of training, maintaining high trainable ranks allows the policy to capture the general patterns of desired behaviors. As training progresses, the number of trainable ranks can be reduced to improve training efficiency, as the policy shifts to incremental refinement. This capability is particularly valuable in scenarios like interactive imitation learning (IL) with diffusion policies.

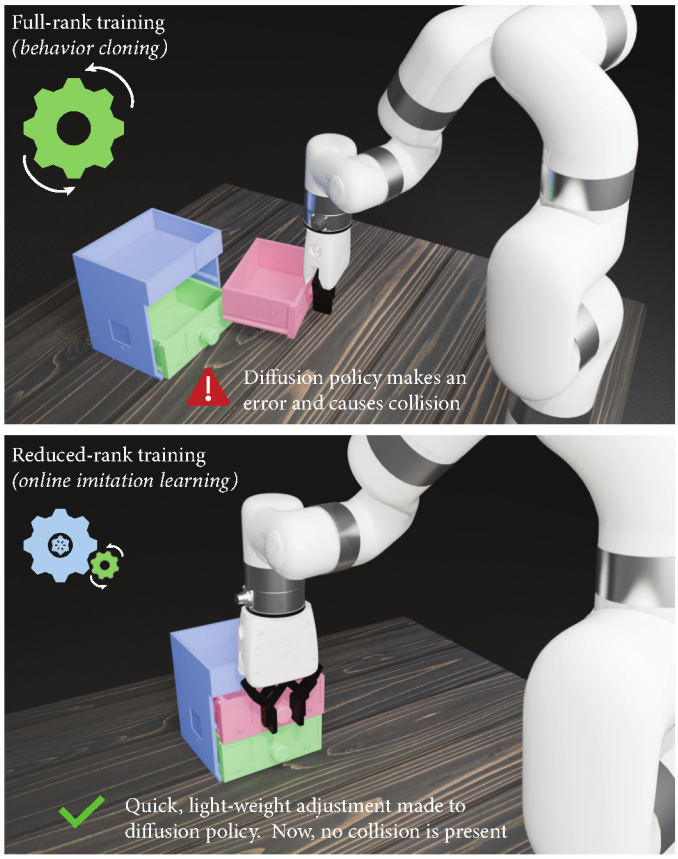

Fig. 1: This paper explores balancing overparameterization and training efficiency in diffusion policies by dynamically adjusting the frozen and trainable portions of weight matrices. In the top section of the figure, the learner, trained offline via behavior cloning with full-rank training, attempts to insert the upper drawer box into the container but fails due to collisions with both the container and the lower drawer box. In the bottom section, after efficient online adaptation with reduced trainable ranks, the learner efficiently improves its performance, successfully completing the task.

Before the adoption of diffusion policies, interactive IL methods typically used simple and compact network architectures as policy backbones. These methods were developed to address the sample inefficiency of behavior cloning (BC)

[41, 49], and often involve an offline bootstrapping stage for initial training, followed by an online adaptation stage where experts provide corrective interventions to refine the policy. However, directly extending these methods to diffusion policies is impractical due to their significantly larger number of trainable parameters, which is often an order of magnitude greater than those of compact networks and results in substantially increased training times. This challenge undermines the feasibility of online interactive IL with diffusion policies.

Typically, diffusion policies are trained offline via BC, where a large dataset of demonstrations is collected, and training occurs in isolation. However, when these policies underperform, the expert must collect additional demonstrations targeting the challenging trajectories, provide corrective demonstrations, and retrain the policy iteratively in an offline manner. This process is both inefficient and unintuitive, as the expert often has limited insight into the trajectories where the policy struggles and may find it difficult to reproduce such challenging scenarios.

To address these limitations, we propose **D**ynamic **R**ank-adjustable **DIF**fusion Policy **T**raining (DRIFT), a framework designed to enable dynamic adjustments of the number of trainable parameters in diffusion policies through reduced-rank training. The framework introduces two key components, which are *rank modulation* that leverages Singular Value Decomposition (SVD) [39] to adjust the proportion of trainable and frozen ranks while maintaining the total rank constant, and *rank scheduler* that dynamically modulates the number of trainable ranks during training using a decay function.

To demonstrate and evaluate the effectiveness of DRIFT, we instantiate and implement it into DRIFT-DAgger, an expert-gated interactive IL method that incorporates reduced-rank training. As shown in Fig. 2, DRIFT-DAgger uses low-rank component to speed up training of diffusion policies. By freezing a significant portion of the ranks during online adaptation, DRIFT-DAgger reduces training time, making online interactive IL with diffusion policies more practical.

Despite being inspired by existing parameter-efficient fine-tuning methods, DRIFT-DAgger with rank modulation and rank scheduler is specifically designed for training diffusion policies from scratch and dynamically adjusting the trainable ranks, avoiding the need to reinitialize and inject low-rank components during training. This design enhances stability and reduces the time for forward passes during each training batch (§VI-D). Additionally, we perform extensive ablation studies on different variants of rank schedulers (§VI-A), and minimum trainable ranks (§VI-B). By combining diffusion policies with online interactive IL, DRIFT-DAgger improves sample efficiency compared to training diffusion policies with BC (§VI-C). We also validate our methods in real-world scenarios (§VII). Finally, we discuss the limitations and implications of our work (§VIII). Our contributions are as follows:

- We propose DRIFT, a framework for diffusion policies that includes rank modulation and rank scheduler as novel components that exploit the intrinsic low-rank structure of overparameterized models, balancing training efficiency and model performance.
- We instantiate DRIFT into DRIFT-DAgger, an interactive IL algorithm that combines offline bootstrapping with efficient online adaptation, enabling effective integration of expert feedback during the novice policy training.
- We perform extensive experiments to demonstrate that DRIFT-DAgger improves sample efficiency and reduces training time while achieving comparable performance to diffusion policies trained with full rank.
- We provide open-source implementations in Pytorch for the DRIFT framework and DRIFT-DAgger algorithm.[1]

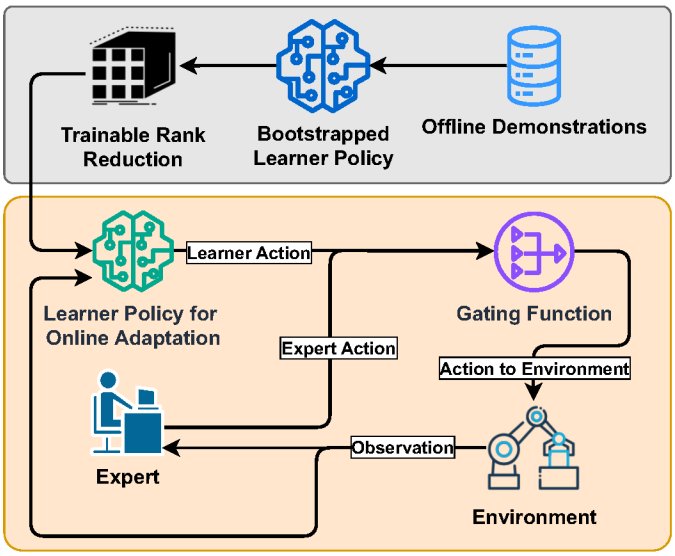

Fig. 2: DRIFT-DAgger combines offline policy bootstrapping with online adaptation. The gating function, following the nomenclature of HG-DAgger [18], refers to expert intervention and demonstration when the learner reaches undesirable states during online adaptation. Compared to BC, DRIFT-DAgger reduces the need for expert labeling while maintaining high performance. The trainable rank reduction accelerates batch training, improving the usability and practicality of online adaptation for large models without sacrificing performance.

II. Background

A. Diffusion Policy Primer

A denoising diffusion probabilistic model (DDPM) [10, 36] consists of a forward process and a reverse process. In the forward process, Gaussian noise is gradually added to the training data, $x_0 \sim p(x_0)$, over T discrete time steps. This process is governed by a predefined noise schedule, β_t, which controls how much noise is added at each step. Mathematically, the forward process is defined as:

$$q(x_{1:T} \mid x_0) := \prod_{t=1}^{T} q(x_t \mid x_{t-1}),$$
$$q(x_t \mid x_{t-1}) := \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t I),$$

[1]https://github.com/Apollo-Lab-Yale/drift_dagger

where $q(x_t \mid x_{t-1})$ is a Gaussian distribution with a mean of $\sqrt{1-\beta_t}x_{t-1}$ and variance β_t. Intuitively, this step progressively adds noise to x_0, such that by the final step x_T, the data is almost entirely noise.

The reverse process aims to undo this noise, step by step, to recover the original data x_0. This is parameterized by a neural network, π_θ, which predicts the noise added to x_t at each step t. Using this prediction, the reverse process reconstructs the data from the noisy input x_t:

$$x_{t-1} \sim p_\theta(x_{t-1} \mid x_t) := \mathcal{N}(x_{t-1}; \mu_k(x_t, \pi_\theta(x_t, t)), \sigma_t^2 I),$$

where $\mu_k(\cdot)$ computes the mean for the denoised data at step $t-1$, and σ_t^2 is a fixed variance term.

In the context of robotics, a diffusion policy [3] adapts the DDPM framework for visuomotor control by treating robot actions as x and conditioning the denoising process on robot observations, such as camera images or sensor data. Specifically, the noise prediction network π_θ takes the current noisy action representation x_t and the observations as inputs and predicts the noise to remove. Architectures like U-Nets [33] or transformers [44] are commonly used for π_θ. Diffusion policies are typically trained offline using BC, where the model learns to mimic expert demonstrations.

B. Ranks in Diffusion Models

The rank of a matrix, defined as the maximum number of linearly independent rows or columns [39], is closely tied to the expressiveness and representational power of a model. For example, in linear models, the rank of the weight matrix determines the dimensionality of the feature space that the model can effectively capture. Weight matrices with low ranks often correspond to models with limited capacity but faster training, while those with high ranks indicate overparameterization, which can improve optimization but at the cost of slower training [7].

In the context of a diffusion policy that employs a U-Net with one-dimensional convolutional blocks as its network backbone, a weight matrix for each convolutional block $W \in \mathbb{R}^{m \times n}$ can be created by reshaping a corresponding weight tensor $W_{\text{conv}} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k}$ via

$$W = \texttt{reshape}(W_{\text{conv}}, (m, n)),$$

where C_{out} is the number of output channels, C_{in} is the number of input channels, and k is the kernel size. The reshaping can be performed by setting $m = C_{\text{out}} * k$ and $n = C_{\text{in}}$ or other equivalent view transformations.

The highest possible rank $r_{\max}$ of this weight matrix is bounded by:

$$r_{\max} \leq \min(m, n).$$

C. Problem Statement

In this work, we investigate diffusion policies that use a U-Net backbone composed of one-dimensional convolutional blocks, as introduced above. For each convolutional block with weight W and highest possible rank $r_{\max}$ in a diffusion policy π_θ, we aim to enable dynamic adjustment of the rank r of a trainable segment of the weight matrix, W_{train}, for any r integer satisfying $1 \leq r \leq r_{\max}$. We assume all weight matrices W throughout the network π_θ will vary uniformly based on r. Importantly, r should remain adjustable throughout the learning process without introducing instability or computational overhead.

III. Related Works

A. Overparameterization and Intrinsic Rank

Overparameterization, where models have more parameters than necessary to fit the training data, is a key factor behind the success of modern machine learning [21, 16]. Large models such as diffusion models [10] and transformers [44] excel in tasks like image synthesis [32], robotic manipulation [3], and language generation [53]. Although overparameterized models offer impressive performance, their size also poses significant challenges for training and fine-tuning due to high computational and memory requirements.

To tackle these challenges, researchers have observed that overparameterized models often reside in a low-dimensional subspace [1, 24, 47]. Pre-deep learning approaches like dynamical low-rank approximation [20] assume a derivable target matrix, unsuitable for deep learning where the target weight matrix is unknown. More modern techniques like Low-Rank Adaptation (LoRA) [13, 6] fine-tune a small low-rank adapter while keeping the main model frozen. Although LoRA and its variants like DyLoRA [43] and QLoRA [6] effectively reduces computational costs, it is primarily suited for fine-tuning pre-trained models and is less practical for training models from scratch due to its fixed low-rank structure [27].

For LoRA, the need to merge and re-inject adapters during rank adjustments and the increased parameters during forward pass can destabilize training and increase computational overhead. In contrast, the DRIFT framework dynamically adjusts trainable ranks without adding new parameters or introducing instability. By using SVD to partition weight matrices into trainable and frozen components, DRIFT maintains stability and efficiency, making it well-suited for training overparameterized diffusion policies from scratch.

B. Imitation Learning and Diffusion Policy

Imitation Learning (IL) is a widely studied policy learning paradigm applied to various robotic scenarios. IL involves collecting demonstration data from an expert and training a neural network using supervised learning techniques [48]. Before the emergence of diffusion policies, IL research focused on improving sample efficiency and mitigating compounding errors through strategic sample collection and labeling [37]. Ross et al. [34] first address these challenges with an iterative, interactive IL method. This approach collects additional demonstration rollouts using a suboptimal policy and refines the trajectories with corrections provided by an expert. Building on this work, expert-gated [17, 40] and learner-gated [11, 12] methods allow experts or learners to dynamically take or hand over control during online rollout collection, which further improves data efficiency.

These methods primarily rely on interactive demonstration strategies and typically utilize simple neural network architectures, such as Multi-Layer Perceptrons (MLPs) [15, 28, 54] or Long Short-Term Memory (LSTM) networks [2, 14, 46]. Historically, these interactive IL methods often employ shallow and small MLPs or LSTM, which are constrained by their relatively small number of parameters, limiting the performance of the learned policies.

Diffusion policies [3] shift the focus of IL research to leveraging the representational power of overparameterized models. Inspired by generative models [10, 36], diffusion policies use large networks to achieve strong performance in various tasks. However, the computational demands of these models create challenges for both training and inference. Few existing works attempt to integrate interactive IL with diffusion models [23, 52]. For example, Lee and Kuo [23] leverage diffusion loss to better handle multimodality; however, this work focuses on a robot-gated interactive approach rather than an expert-gated one and does not contain real-world experiments. Similarly, while Zhang et al. [52] employ diffusion as a policy representation, the primary innovation in this work lies in using diffusion for data augmentation rather than improving the interactive learning process. Notably, neither approach addresses the critical issue of reducing batch training time, which is essential for making online interactive learning with large models more practical and usable. Recent efforts to accelerate diffusion policies focus on inference through techniques like distillation [31, 45], but no existing work focuses on improving training efficiency. As a result, diffusion policy research remains largely confined to offline training scenarios [38, 42, 50].

IV. DRIFT Framework

A. Overview

The DRIFT framework is designed to dynamically adjust trainable ranks in a diffusion policy, allowing the number of trainable parameters to change throughout the training process. This flexibility enables efficient training from scratch by leveraging the intrinsic low-rank structure of overparameterized models. As covered in §II-C, the rank adjustment process must ensure training stability and avoid introducing additional computational overhead. These considerations are critical for training from scratch but are often overlooked by existing methods like LoRA [13], which are primarily designed for fine-tuning. Applying methods like LoRA for dynamic rank adjustment during training from scratch can result in higher computational time for forward pass and instability due to the need for merging and re-injecting newly initialized low-rank adapters, which disrupts the training process.

To achieve dynamic trainable rank adjustment while maintaining training stability, we propose *rank modulation*. Rank modulation uses the Singular Value Decomposition (SVD) [39] to partition ranks into trainable and frozen sections. This approach avoids introducing new parameters, ensures that computational costs for the backward pass decrease as the trainable ranks are reduced, and maintains a constant computational cost for the forward pass.

In addition to rank modulation, the framework can incorporate a *rank scheduler* to coordinate the dynamic adjustment of trainable ranks. While rank modulation facilitates the adjustment itself, the rank scheduler determines how the trainable ranks may automatically evolve during training. The rank scheduler uses a decay function that calculates the current number of trainable ranks based on the training epoch, the maximum rank, and the desired terminal rank of the policy.

The rank modulation and rank scheduler components can work together to enable efficient training of diffusion policies by dynamically balancing representational power and computational efficiency.

B. Rank Modulation

Rank modulation takes inspiration from LoRA [13], which employs an adapter with small trainable ranks during fine-tuning. LoRA achieves this by injecting additional low-rank weight matrices into the network layers, allowing only these matrices to be updated during backpropagation. For the one-dimensional convolution blocks in a U-Net architecture used by diffusion models, LoRA would replace the original convolutional layer with:

$$\text{Conv}_{\text{LoRA}}(x) = W_{\text{conv}} \circledast x + \alpha((W_{\text{up}} \times W_{\text{down}}) \circledast x),$$

where $\circledast$ denotes convolution, W_{conv} is the original convolution weight tensor of shape $(C_{\text{out}}, C_{\text{in}}, k)$, with C_{out} and C_{in} denoting the output and input channels, and k is the kernel size. Two low-rank matrices, $W_{\text{down}} \in \mathbb{R}^{r \times C_{\text{in}} \times k}$ and $W_{\text{up}} \in \mathbb{R}^{C_{\text{out}} \times r \times k}$, are introduced, where $r \ll C_{\text{in}}$. A scaling factor α further controls the magnitude of the low-rank update. During backpropagation, gradients are computed only for W_{down} and W_{up}, thus lowering the number of trainable ranks.

Despite these benefits, LoRA has several limitations when applied to IL that trained from scratch. Since LoRA is essentially an approximation of the full-rank weight, it relies on the main weight W_{conv} being thoroughly pre-trained. Otherwise, the low-rank approximation may limit the representational power of the model and prevent it from fully benefiting from overparameterization. Furthermore, injecting LoRA blocks adds complexity to the forward pass. Due to the merging of W_{up} and W_{down}, the additional convolution with LoRA blocks results in a time complexity of $O(C_{\text{out}} \times C_{\text{in}} \times r \times k)$, which increases computational overhead proportional to the rank r compared to the time complexity of the original convolution $O(C_{\text{out}} \times C_{\text{in}} \times k)$. While this computational overhead is often negligible when fine-tuning a pre-trained model with a rank of less than 4 [13], training a model from scratch requires low-rank adapters with significantly higher number of trainable ranks to effectively leverage both overparameterization and low-rank structure (as we will demonstrate in §VI-B). This increase in trainable ranks amplifies the computational cost associated with r, making it a critical consideration in such scenarios.

Finally, if LoRA is used with a dynamic rank scheduler (discussed in §IV-C), new LoRA blocks must be injected each time the rank changes, introducing freshly initialized parameters and destabilizing training. Consequently, repeatedly merging and reinjecting LoRA blocks is inefficient when the trainable rank is adjusted on the fly.

To address these limitations, we propose rank modulation, which leverages an SVD structure to decompose weight matrices into components designated as either frozen or trainable ranks. More specifically, consider a weight matrix $W \in \mathbb{R}^{m \times n}$ that can be created from a corresponding weight tensor $W_{\text{conv}} \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k}$ for a one-dimensional convolutional layer by reshaping, e.g., $C_{\text{out}} * k$ becomes m and C_{in} becomes n (as covered in §II-B). This matrix can be refactored into three matrices using the SVD:

$$W = U \, \Sigma \, V^T,$$

where $U \in \mathbb{R}^{m \times m}$, $\Sigma \in \mathbb{R}^{m \times n}$, $V \in \mathbb{R}^{n \times n}$. U and V are orthonormal matrices that represent rotations or reflections, while Σ is a diagonal matrix containing scaling factors. We can further split U, Σ, and V at a specified rank r to partition trainable and frozen part of each matrix:

$$\begin{aligned} U &= \begin{bmatrix} U_{\text{train}} & U_{\text{frozen}} \end{bmatrix}, \\ \Sigma &= \begin{bmatrix} \Sigma_{\text{train}} & 0_{r \times (n-r)} \\ 0_{(m-r) \times r} & \Sigma_{\text{frozen}} \end{bmatrix}, \\ V &= \begin{bmatrix} V_{\text{train}} & V_{\text{frozen}} \end{bmatrix}. \end{aligned}$$

Accordingly, we represent trainable weight W_{train} and frozen weight W_{frozen} as:

$$\begin{aligned} W_{\text{train}} &= U_{\text{train}} \, \Sigma_{\text{train}} \, V_{\text{train}}^T, \\ W_{\text{frozen}} &= U_{\text{frozen}} \, \Sigma_{\text{frozen}} \, V_{\text{frozen}}^T, \end{aligned}$$

where Σ_{frozen} holds smaller singular values than Σ_{train}. During training, $\{U_{\text{train}}, \Sigma_{\text{train}}, V_{\text{train}}\}$ are the only parameters that receive gradient updates (rank-r subspace), while $\{U_{\text{frozen}}, \Sigma_{\text{frozen}}, V_{\text{frozen}}\}$ remain fixed.

The procedure described above, converting W_{conv} into W_{train} and W_{frozen} using SVD and partitioning, is performed at the start of each epoch if the number of trainable ranks has changed since the previous epoch. This procedure also reorthonormalizes the U and $V^\top$ matrices. While it is possible to reorthonormalize the U and $V^\top$ matrices more frequently (such as after each gradient update) using QR decomposition, preliminary experiments showed that this introduced significant computational overhead without measurable performance gains (see Appendix §IX-A). Therefore, our implementation defaults to reorthonormalizing only via full SVD at the start of an epoch whenever the number of trainable ranks changes.

Unlike LoRA, rank modulation performs a single convolution using the full $W = W_{\text{train}} + W_{\text{frozen}}$ via another simple view transformation in memory:

$$W_{\text{conv}} = \texttt{reshape}(W, (C_{\text{out}}, C_{\text{in}}, k)).$$

Hence, the forward time complexity remains the same as a standard convolution. Because no additional new parameters are introduced during the training process, rank modulation can also preserve stable updates even when the number of trainable ranks changes.

C. Rank Scheduler

Building on rank modulation, which dynamically adjusts the number of trainable ranks while maintaining stable training, we introduce a rank scheduler to further exploit the low-rank structure. The rank scheduler, inspired by the noise scheduler in diffusion models that dynamically adjusts the added noise [10], is designed to improve training efficiency without compromising performance.

The rank scheduler uses a decay function to compute the current number of trainable ranks r_i based on the current training epoch index i, and the maximum and minimum trainable ranks, r_{max} and r_{min}. Once r_i is determined, the trainable ranks are adjusted depending on the low-rank adapters. For instance, with LoRA, this process involves merging the current LoRA blocks and reinstantiating new blocks with the updated trainable ranks. In the case of rank modulation, this process entails computing the SVD of W to produce updated W_{train} and W_{frozen} matrices.

In this work, we implement and evaluate four decay functions, which are linear, cosine, sigmoid, and exponential:

$$r_{\text{linear}} = \left\lfloor r_{\text{max}} - (r_{\text{max}} - r_{\text{min}}) \times \left(\frac{i}{T} \right) \right\rfloor$$

$$r_{\text{cosine}} = \left\lfloor r_{\text{min}} + 0.5 \times (r_{\text{max}} - r_{\text{min}}) \times \left(1 + \cos \left(\pi \times \frac{i}{T} \right) \right) \right\rfloor$$

$$r_{\text{sig}} = \left\lfloor r_{\text{max}} - \frac{(r_{\text{max}} - r_{\text{min}})}{1 + e^{-\tau \times (i - t_m)}} \right\rfloor$$

$$r_{\text{exp}} = \left\lfloor r_{\text{min}} + (r_{\text{max}} - r_{\text{min}}) \times e^{-\tau \times i} \right\rfloor$$

where i, T, and t_m are the current, total, and midpoint of the number of training epochs, respectively, τ denotes steepness, and $\lfloor \cdot \rfloor$ is the floor function.

V. DRIFT-DAgger

DRIFT-DAgger combines the sample efficiency of interactive IL with the computational efficiency of low-rank training methods, making it well-suited for training large policies interactively.

Algorithm 1 outlines the DRIFT-DAgger procedure. Similar to previous interactive IL methods, DRIFT-DAgger consists of an offline bootstrapping stage followed by an online adaptation stage. The process begins with an initial policy π_{N_0}, parameterized by a neural network that can adjust the number of trainable ranks. Although DRIFT-DAgger is proposed as an instantiation of the DRIFT framework, the adjustment of trainable ranks can be achieved through any kind of low-rank adapters other than the rank modulation proposed in §IV-B, such as LoRA.

In the offline bootstrapping stage, DRIFT-DAgger trains the policy π_{N_i} on an offline dataset $\mathcal{D}_B$ over several epochs i using BC, similar to prior interactive IL methods. However, unlike

these methods, DRIFT-DAgger optionally employs a rank scheduler that gradually reduces the number of trainable ranks during training. The rank scheduler uses a decay function based on the epoch index i, along with the highest possible ranks ($r_{\max}$) and terminal trainable ranks ($r_{\min}$) for the policy network. This approach reduces computational costs while maintaining performance. Details of the rank scheduler are presented in §IV-C.

If the rank scheduler is not used, the number of trainable ranks is set fixed at r_{min} after offline bootstrapping and before transitioning to the online adaptation stage. At the end of offline bootstrapping, the offline dataset $\mathcal{D}_B$ is merged into a global dataset $\mathcal{D}$ for further use in online adaptation.

During the online adaptation stage, the learner policy interacts with the environment through rollouts. At each iteration j, the learner executes a rollout in the environment. If the expert policy π_{exp} detects that the learner has deviated from the desired trajectory, the expert intervenes, taking control to provide corrective demonstrations. The expert can be a human teleoperator, an algorithm, or another neural network. These demonstrations are recorded in a dataset specific to the current rollout $\mathcal{D}_j$. After each rollout, $\mathcal{D}_j$ is merged into the global dataset $\mathcal{D}$, and the learner policy $\pi_{N_{\mathcal{I}+j}}$ is updated using the expanded dataset $\mathcal{D}$.

The full procedure of DRIFT-DAgger leverages low-rank training and online interaction to achieve better sample and training efficiency.

Algorithm 1: DRIFT-DAgger

procedure DRIFT-DAgger(π_{exp}, π_{N_0}, $\mathcal{D}_B$)
  for offline epoch $i = 1, 2, \cdots, \mathcal{I}$ **do**
    train π_{N_i} on offline dataset $\mathcal{D}_B$
    if use rank scheduler **then**
      $r_i = \texttt{Decay Function}(i, r_{\min}, r_{\max})$
      $\pi_{N_i} = \texttt{Rank Reduction}(r_i, \pi_{N_i})$
    if not use rank scheduler **then**
      $\pi_{N_{\mathcal{I}}} = \texttt{Rank Reduction}(r_{\min}, \pi_{N_{\mathcal{I}}})$
  $\mathcal{D} \leftarrow \mathcal{D}_B$
  for online iteration $j = 1, 2, \cdots, \mathcal{J}$ **do**
    for timestep $t \in T$ of online rollout j **do**
      if π_{exp} takes control **then**
        $observation \leftarrow \texttt{rollout}_j^t$
        $action \leftarrow \pi_{exp}(observation)$
        $\mathcal{D}_j \leftarrow (observation, action)$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}_j$
    Train $\pi_{N_{\mathcal{I}+j}}$ on $\mathcal{D}$
  return $\pi_{N_{\mathcal{I}+\mathcal{J}}}$

VI. Simulation Evaluation

We evaluate the proposed DRIFT framework, instantiated in the DRIFT-DAgger algorithm, through extensive simulation experiments and ablation studies. All experiments are conducted using the PyTorch framework [30], with the UNet-based diffusion policies that enable RGB perception following

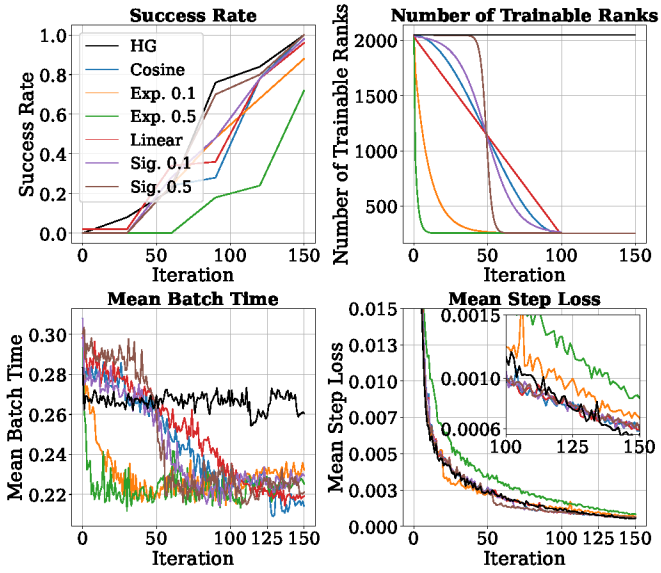

Fig. 3: Experimental results of DRIFT-DAgger with different decay functions for the rank scheduler. We use HG-DAgger (HG) as a baseline for comparison.

TABLE I: Summary of experimental results on mean batch training time (MBT) and success rate with different rank decay functions for DRIFT-DAgger.

Function	Success Rate	MBT (Offline)	MBT (Online)	MBT (All Stages)
HG	**1.0**	0.27	0.27	0.27
Linear	0.96	0.26	0.22	0.25
Cosine	**1.0**	0.26	0.22	0.25
Exp. 0.1	0.88	0.23	0.23	0.23
Exp. 0.5	0.72	**0.22**	**0.22**	**0.22**
Sig. 0.1	0.98	0.25	0.23	0.24
Sig. 0.5	**1.0**	0.26	0.22	0.24

the specifications from Chi et al. [3]. The batch size and learning rate is set to 256 and 10^{-4} for all experiments. We use Adam [19] as the optimizer. Training is performed on a desktop PC with an AMD PRO 5975WX CPU, 4090 GPU, and 128GB RAM. To ensure a fair comparison with interactive methods like HG-DAgger [18] and DRIFT-DAgger, we implement BC with an incremental dataset during the online phase, similar to the interactive loop of HG-DAgger and DRIFT-DAgger.

A. Decay Functions

To identify the scheduling strategy that best exploits the benefits of reduced-rank training while balancing the trade-off between training time and policy performance, we evaluate six variants of four decay functions for the rank scheduler. These functions dynamically adjust the number of trainable ranks as training progresses. The decay functions considered include linear, cosine, exponential, and sigmoid, as covered in §IV-C,

with steepness parameters τ set to 0.1 and 0.5 for exponential and sigmoid.

We use DRIFT-DAgger with rank modulation and rank scheduler for this ablation study. We set r_{min} to 256 for all DRIFT-DAgger variants. Since both BC and HG-DAgger employ full-rank training, they should exhibit similar batch training times. Given that HG-DAgger outperforms BC in terms of sample efficiency[17], we use HG-DAgger as the baseline for full-rank training methods.

To assess performance and training efficiency across different decay functions, we evaluate the policy's success rate, with higher values indicating better task completion. We also track the step loss during training as a measure of convergence. To assess training efficiency, we record the mean batch training time per epoch, separately for the offline stage, online stage, and their combination, interpreting it alongside the success rate and step loss.

The ablation study is conducted on a pick-and-place scenario from the Manipulation with Viewpoint Selection (MVS) tasks [42]. The pick-and-place scenario, as illustrated in Fig. 5, requires the agent to pick up a green cube and place it in a red region. All MVS tasks involve two UFactory xArm7 robots[2] mounted on linear motors, where one arm has a gripper and the other is equipped with a camera. Mounted on one end effector, the camera enables active perception, working in synergy with the gripper on the other end effector to cooperatively execute the manipulation task. The state-action space for all MVS tasks is a reduced end-effector space for the dual-arm system, with automatically computed camera orientation using an additional Inverse Kinematics (IK) objective.

The learning process uses 100 offline demonstration rollouts, 100 offline bootstrapping epochs, and 50 online iterations. We plot the experimental results by combining the number of offline epochs and online iterations, resulting in a total of 150 iterations.

The results of this experiment are presented in Fig. 3 and summarized in Table I. While all decay functions reduce batch training time compared to full-rank training represented by HG-DAgger, some decay functions lead to decreased policy performance, as indicated by the success rates in Fig. 3 and Table I. Notably, the exponential decay functions, due to their aggressive reduction of trainable ranks, underperform relative to the other variants, despite yielding the lowest mean batch training time.

The linear decay function, while offering near-perfect policy performance, results in the highest training time among all variants, suggesting that it is less effective than the sigmoid decay functions in balancing training time with performance. We observe that the sigmoid functions, particularly with a steep decay parameter $\tau = 0.5$, strike the best balance between training time and policy performance. These functions maintain a high number of trainable ranks during the early training phase, allowing overparameterization to effectively minimize approximation error, as shown in the loss plot in Fig. 3. This facilitates more efficient learning of the predominant behavior during the early stage of training, while still preserving the flexibility required for online adaptation.

[2]https://www.ufactory.us/product/ufactory-xarm-7

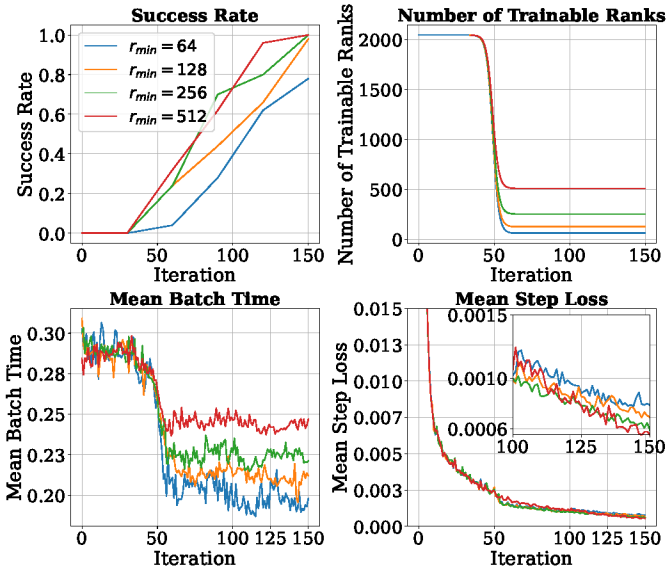

Fig. 4: Experimental results of DRIFT-DAgger with different terminal ranks r_{min}.

TABLE II: Summary of experimental results on mean batch training time (MBT) and success rate with different values for terminal ranks for DRIFT-DAgger.

r_{min}	Success Rate	MBT (Offline)	MBT (Online)	MBT (All Stages)
64	0.78	**0.24**	**0.19**	**0.22**
128	0.98	0.25	0.21	0.23
256	**1.0**	0.26	0.22	0.24
512	**1.0**	0.27	0.24	0.26

B. *Terminal Rank*

To explore the effect of different terminal ranks for the DRIFT framework, we conduct an ablation study by varying the terminal rank r_{min} in DRIFT-DAgger with rank modulation and rank scheduler. We use the same experimental task, setup, and evaluation metrics as §VI-A, and use sigmoid decay function with τ set to 0.5 for rank scheduler. The terminal rank r_{min} is set to 64, 128, 256, and 512 for comparison.

As shown in Fig. 4 and summarized in Table II, decreasing the terminal ranks reduces training time but also impacts policy performance, as reflected in the success rate. This performance degradation occurs due to the diminished representational power of the model when the number of trainable ranks is reduced. Compared to reduced-rank methods for fine-tuning, such as [13], where fine-tuning with an adapter requires only 4 trainable ranks, DRIFT-DAgger with reduced-rank training from scratch necessitates a significantly higher number of trainable ranks, given that there is a noticeable performance

drop when r_{min} is set to 64. This is to fully leverage the benefits of both better representation power from overparameterization and improved computational efficiency from intrinsic low ranks, as fine-tuning with extremely small adapters does not capture the full representational potential of the model.

Conversely, setting r_{min} too high, such as 512, offers no clear benefit in terms of policy performance, as the model already achieves a perfect success rate. The approximation error, as indicated by the loss curve in Fig. 4, also shows negligible differences between r_{min} values of 256 and 512. We also run r_{min} sweep for another MVS task, and the results follow similar trends as the pick-and-place scenario that $r_{\text{min}} = 256$ provides the best balance of efficiency and performance (see Appendix §IX-B).

TABLE III: The configurations of DRIFT-DAgger for all simulation and real-world tasks. For simulation, we use other neural network policies trained with BC as experts, and compare the cosine similarity of the expert action and learner action with the threshold to determine whether the expert take over or not.

	Task	Offline Rollouts	Boot. Epochs	Online Iterations	Cos. Sim. Threshold
Sim.	Robo-L	300	100	100	0.94
	Robo-C	275	100	100	0.95
	MVS-M	75	35	100	0.99
	MVS-PnP	100	100	100	0.99
Real	Real-BS	150	150	50	-
	Real-DA	200	200	50	-
	Real-DI	100	150	50	-

C. Benchmark Comparison

We perform a benchmark comparison in four environments: two from the robosuite [55] and two from the Manipulation with Viewpoint Selection (MVS) tasks [42]. Fig. 5 provides illustrations of the four environments. The robosuite environments, Lift and Can, involve a Panda arm performing manipulation tasks, such as lifting a red cube and placing a can into a category. The state-action spaces for the robosuite tasks are the end-effector space with quaternions for rotation. The MVS tasks include opening a microwave, and the same pick-and-place scenario we used for previous ablation studies.

The methods we evaluate in this benchmark comparison include Behavior Cloning (BC), HG-DAgger, and three variants of DRIFT-DAgger: one that uses LoRA, one that uses LoRA with rank scheduler, and one that uses rank modulation and rank scheduler. For methods utilizing rank scheduler, we apply the sigmoid decay function with steepness τ set to 0.5. We set r_{max} and r_{min} to 2048 and 256, respectively, based on the maximum rank of the diffusion policy and prior ablation study on the terminal rank. For all methods that use LoRA, we set the scaling factor α to 1.0. Experimental parameters related to the interactive mechanism, including the number of offline rollouts in $\mathcal{D}_B$, bootstrapping epochs $\mathcal{I}$, and online iterations $\mathcal{J}$, are provided in Table III.

During the online iterations, we save checkpoints every 20 iterations. Each checkpoint undergoes evaluation with 50 rollouts, and the success rate, mean and standard deviation of task duration, the number of expert labels, and cumulative training time are recorded as metrics for comparison. The success rate is used as the primary performance metric. The mean and standard deviation of task duration reflect how consistent and certain a trained policy is in completing a given task. A lower mean and standard deviation of task duration suggest that the policy is well-trained and converges better to the desired behavior. The number of expert labels, when considered alongside the other metrics, provides insight into the sample efficiency of a specific training method. For example, at the same level of success rate, a lower number of expert labels indicates better sample efficiency. The cumulative training time is for reflecting the training efficiency.

To fairly and efficiently evaluate different methods, for each scenario, we first train an expert policy using human-collected data from Mandlekar et al. [29] and Sun et al. [42] for robosuite and MVS tasks, respectively. The expert policy performs interventions when the cosine similarity between the learner actions and expert actions falls below a threshold, as detailed in Table III. The thresholds are computed based on the mean cosine similarity between consecutive steps in the expert training datasets.

As shown in Fig. 5 and Table IV, DRIFT-DAgger variants demonstrate a pronounced reduction in cumulative training time compared to BC and HG-DAgger. Additionally, all interactive IL methods exhibit superior expert sample efficiency compared to BC, as evidenced by higher success rates with respect to the number of expert labels. An exception is observed in the DRIFT-DAgger variant with LoRA and rank scheduler, where the merging and re-injection of LoRA adapters destabilize training due to the initialization of new trainable parameters. In contrast, the DRIFT-DAgger variant that instantiates LoRA adapters only once during the transition to the online phase, or the variant that combines rank modulation and rank scheduler, achieve performance and sample efficiency comparable to HG-DAgger, which consistently uses full ranks instead of reduced ranks. The benefits of interactive IL are more pronounced in tasks with longer durations. Furthermore, the policies learned with DRIFT-DAgger using the combination of rank modulation and rank scheduler, exhibit good stability and better convergence to the desired behavior of the task, as indicated by the relatively lower standard deviation in task duration.

D. Batch Training Time

To better understand and demonstrate the improved training efficiency of DRIFT-DAgger, we conduct an ablation study on the mean batch training time across all stages. The methods evaluated in this ablation study include HG-DAgger and three DRIFT-DAgger variants: one using LoRA, one combining LoRA with a rank scheduler, and one employing rank mod-

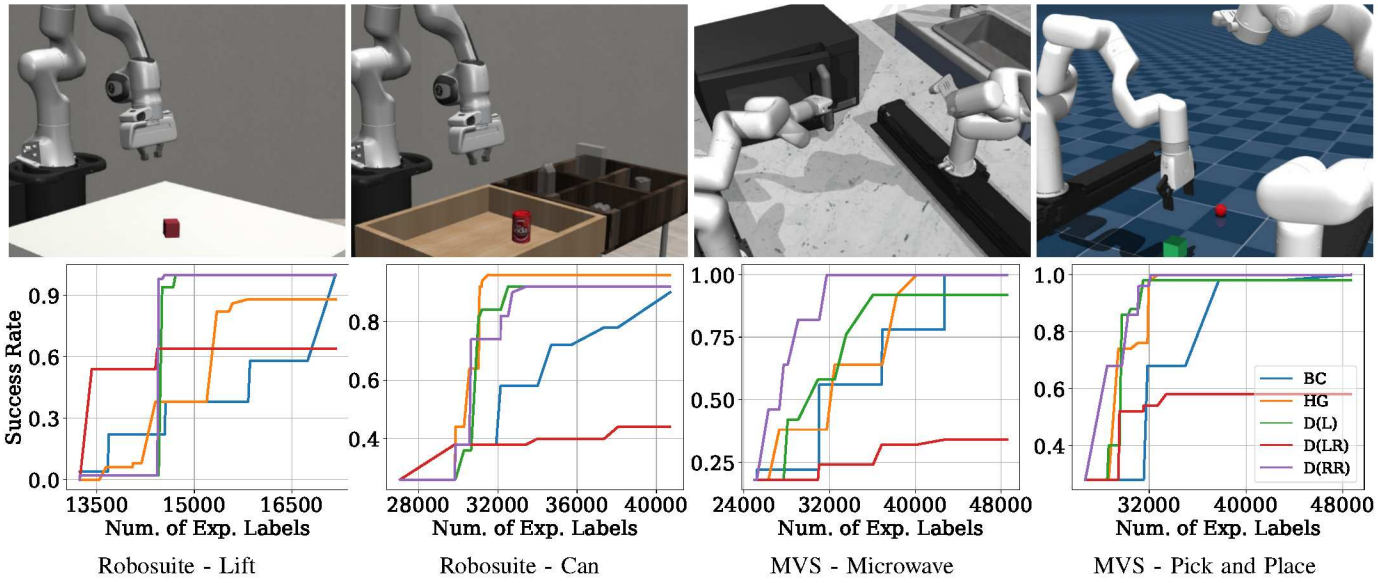

Fig. 5: The upper row shows the simulation scenarios from robosuite and Manipulation with Viewpoint Selection (MVS) tasks. The lower row shows the plots of success rate with respect to the number of expert labels. HG, D(L), D(LR), and D(RR) represent HG-DAgger, DRIFT-DAgger with LoRA adapters that are only instantiated with r_{min} when switching to online mode, DRIFT-DAgger with LoRA and rank scheduler, and DRIFT-DAgger with rank modulation and rank scheduler.

TABLE IV: Summary of experimental results from simulation scenarios. The metrics include success rate (SR), mean and standard deviation of task duration (MSD), number of expert labels (NEL), and cumulative training time (CT). CT is measured in hours, MSD is measured in steps and at the scale of $\times 10^2$, and NEL is at the scale of $\times 10^4$

	Robosuite - Lift				Robosuite - Can				MVS - Microwave				MVS - Pick and Place			
	SR	MSD	NEL	CT	SR	MSD	NEL	CT	SR	MSD	NEL	CT	SR	MSD	NEL	CT
Expert	1.00	0.43 ± 0.03	-	-	0.98	1.17 ± 0.56	-	-	1.00	3.00 ± 0.29	-	-	0.92	3.03 ± 0.69	-	-
BC	**1.00**	0.58 ± 0.61	1.71	1.66	0.90	1.52 ± 1.19	4.06	3.56	**1.00**	3.17 ± 0.75	4.85	2.66	**1.00**	2.61 ± 0.23	4.86	3.76
HG	0.88	1.68 ± 1.40	1.58	1.62	**0.96**	$\mathbf{1.20 \pm 0.80}$	**3.15**	3.30	**1.00**	3.43 ± 0.93	4.01	2.41	**1.00**	$\mathbf{2.54 \pm 0.23}$	3.26	3.30
D(L)	**1.00**	0.73 ± 0.61	1.50	1.48	0.92	1.33 ± 1.08	3.25	3.07	0.92	3.37 ± 0.78	3.60	2.03	0.98	2.60 ± 0.40	**3.14**	3.01
D(LR)	0.54	2.76 ± 2.17	1.47	1.47	0.44	3.61 ± 1.58	3.80	3.20	0.34	4.73 ± 0.42	4.85	2.34	0.58	3.54 ± 0.84	3.49	3.10
D(RR)	**1.00**	$\mathbf{0.50 \pm 0.21}$	**1.46**	**1.41**	0.92	1.42 ± 1.09	3.34	**2.97**	**1.00**	$\mathbf{3.16 \pm 0.60}$	**3.17**	**1.84**	**1.00**	2.73 ± 0.50	3.21	**2.91**

ulation alongside rank scheduler. We use HG-DAgger as the baseline for full-rank training.

This ablation study is performed using the MVS pick-and-place scenario, same as §VI-A and §VI-B. We use 100 offline demonstration rollouts, 100 offline bootstrapping epochs, and 50 online iterations for training. We set terminal ranks r_{min} to 256 and use the sigmoid decay function with τ set to 0.5 for DRIFT-DAgger variants applied.

The results are presented in Fig. 6 and Table V. We observe that the DRIFT-DAgger variant with fixed-rank LoRA and the variant with rank modulation and rank scheduler both achieve success rates comparable to the full-rank baseline, HG-DAgger. When considering batch training time, the variant with rank modulation and rank scheduler reduces the mean batch training time across all stages by 11%, demonstrating improved training efficiency without sacrificing performance. Specifically, during the online training stage, this variant achieves an 18% reduction in batch training time compared to the full-rank baseline.

In contrast, the DRIFT-DAgger variant with LoRA and rank scheduler also shows reduced training time. However, the success rate significantly drops compared to HG-DAgger. This decline is attributed to the instability caused by merging and re-injecting LoRA adapters, which is also reflected in the higher step loss observed for this variant. Additionally, despite the use of a rank scheduler, the batch training time for this variant is slightly higher than that of the variant with rank modulation, likely due to the additional computational overhead introduced by LoRA within the DRIFT framework.

VII. Real-World Evaluation

To further validate the proposed DRIFT framework, we deploy DRIFT-DAgger in three real-world tasks, using a human teleoperator as the expert. The robotic system for these experiments, illustrated in Fig. 7, mirrors the setup used in the

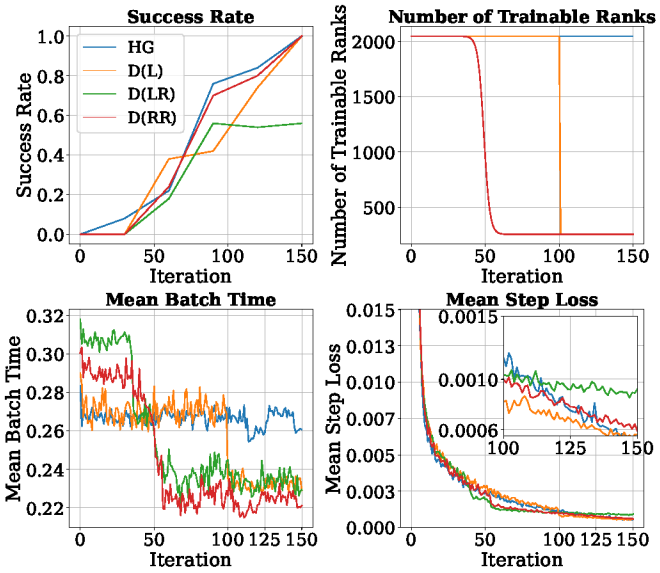

Fig. 6: Experimental results of three DRIFT-DAgger variants with HG-DAgger for demonstrating the reduced batch training time of DRIFT framework compared to full-rank training, while still maintaining equivalent performance.

TABLE V: Summary of experimental results on mean batch training time (MBT) and success rate with full-rank and reduced-rank training methods.

Method	Success Rate	MBT (Offline)	MBT (Online)	MBT (All Stages)
HG	**1.0**	0.27	0.27	0.27
D(L)	**1.0**	0.27	0.23	0.26
D(LR)	0.56	0.27	0.23	0.26
D(RR)	**1.0**	**0.26**	**0.22**	**0.24**

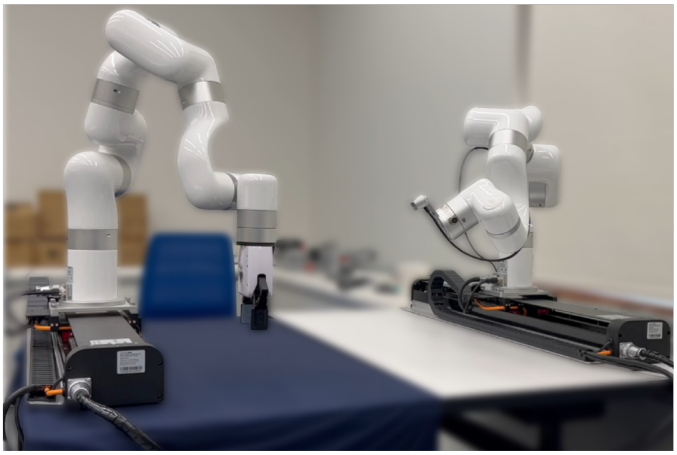

Fig. 7: The 17-DOF robotic system for real-world experiments aligns with the MVS simulation environments. It includes two xArm7 mounted on linear motors, with camera and gripper attached to each end effectors.

simulated MVS tasks. The system comprises two UFactory xArm7 robots mounted on linear motors, with a gripper attached to one arm and a camera to the other. This configuration enables simultaneous manipulation and viewpoint adjustment. The state-action space is a reduced end-effector space, with the orientation of the camera automatically updated via an additional IK objective.

The real-world tasks, depicted in Fig. 8, include block stacking, which involves placing a non-cuboid block on top of another; drawer assembling, which involves inserting two drawer boxes into a drawer container; and drawer interaction, which requires the agent to first get rid of a visual occlusion (cardboard box), grasp a red cube from a cluttered area, place it in the drawer, and then close the drawer. The blocks and drawer components are 3D-printed using models from [22] and [9]. Training configurations for each task are detailed in Table III, and the trained policies from each method are evaluated over 30 rollouts.

We use the success rate, mean and standard deviation of task duration, number of expert labels, and cumulative training time as metrics for the real-world experiments. The cumulative training time only measures the active computation during training, and not includes any other events between two training epochs, such as resetting the robot or recording the demonstration interactively.

Results, summarized in Table VI, show trends consistent with the simulation experiments. All DRIFT-DAgger variants show noticeable reduction in cumulative training time. Interactive IL methods, except the DRIFT-DAgger variant with LoRA and rank scheduler, achieve similar or superior performance to BC, with improved sample efficiency, as indicated by a reduced number of expert labels. Despite using reduced-rank training, the other two DRIFT-DAgger variants perform comparably to HG-DAgger, which trains in a full-rank manner. The advantage of interactive methods becomes more pronounced for longer-duration tasks. For example, in the block stacking task, which has a mean duration of around 0.45 minutes, interactive methods improve sample efficiency by 11.32% to 13.17%. In contrast, for the drawer assembling task, with a mean duration of approximately 1.72 minutes, sample efficiency improves by 14.82% to 17.14%.

These real-world experiments validate that the DRIFT framework is effective in real-world settings, offering reduced training time, improved sample efficiency and robust performance despite employing reduced-rank training.

VIII. Discussion

This work introduces DRIFT, a framework designed to leverage the intrinsic low-rank properties of large diffusion policy models for efficiency while preserving the benefits of overparameterization. To achieve this, we propose rank modulation and rank scheduler, which dynamically adjust trainable ranks using SVD and a decay function. We instantiate DRIFT within an interactive IL algorithm, DRIFT-DAgger, and show this efficacy of this method through extensive

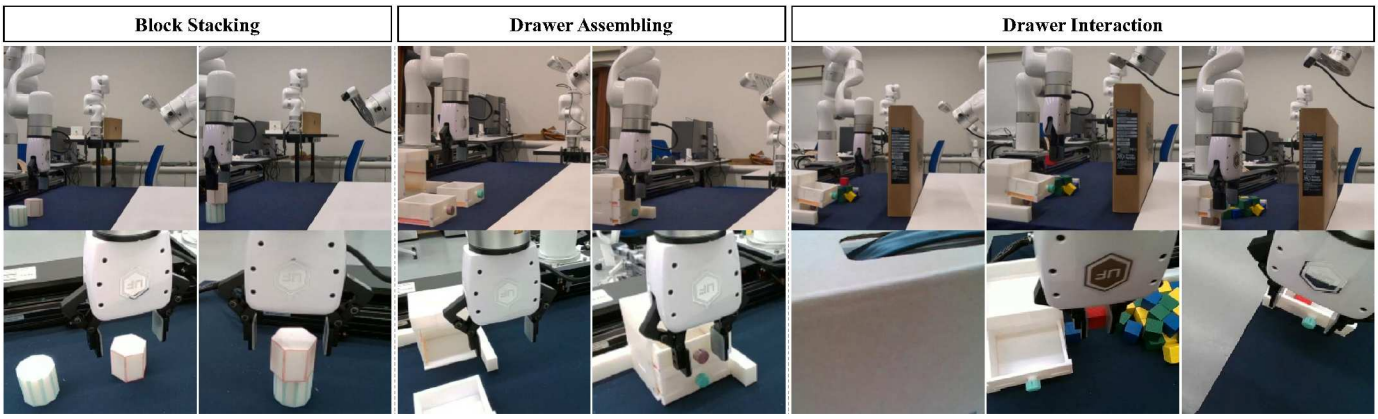

Fig. 8: The images show the tasks for real-world experiments. The upper row and lower row show the process of each task from a third-person perspective and a robot-perception perspective respectively.

TABLE VI: Summary of experimental results from real-world scenarios. The metrics include success rate (SR), mean and standard deviation of task duration (MSD), number of expert labels (NEL), and cumulative training time (CT). CT is measured in hours, MSD measured in minutes, and NEL is at the scale of $\times 10^4$.

	Block Stacking				Drawer Assembling				Drawer Interaction			
	SR	MSD	NEL	CT	SR	MSD	NEL	CT	SR	MSD	NEL	CT
BC	0.97	$\mathbf{0.41 \pm 0.04}$	4.86	4.45	0.40	1.76 ± 0.21	12.08	14.50	0.83	0.94 ± 0.11	4.82	3.99
HG	**1.00**	0.42 ± 0.04	**4.22**	4.35	**0.77**	$\mathbf{1.52 \pm 0.18}$	10.21	14.21	0.90	0.85 ± 0.12	**3.94**	3.87
D(L)	**1.00**	0.43 ± 0.05	4.31	4.19	0.67	1.55 ± 0.20	10.29	13.81	0.87	0.89 ± 0.11	4.04	3.73
D(LR)	0.53	0.58 ± 0.05	4.57	4.23	0.20	2.18 ± 0.24	12.11	14.04	0.37	1.09 ± 0.18	4.53	3.79
D(RR)	**1.00**	0.43 ± 0.05	4.25	**4.03**	0.73	1.58 ± 0.15	**10.01**	**13.28**	**0.93**	$\mathbf{0.84 \pm 0.10}$	4.08	**3.59**

experiments and ablation studies in both simulation and real-world settings. Our results demonstrate that DRIFT-DAgger reduces training time and improves sample efficiency while maintaining performance on par with full-rank policies trained from scratch.

A. Limitations

This work evaluates and demonstrates the DRIFT framework using DRIFT-DAgger as an instantiation within the IL paradigm. However, prior to the adoption of large models, online reinforcement learning (RL) approaches [35, 8] were also a popular area of research. This work does not explore the application of the DRIFT framework in the online RL paradigm. Investigating the potential of DRIFT within online RL could serve as an valuable direction for future research.

Additionally, while we have tested and evaluated various decay functions for the rank scheduler, the current implementation of dynamic rank adjustment in the DRIFT framework follows a monotonic schedule. Although we have conducted ablation studies on decay functions and terminal ranks, the impact of these design choices is likely task-dependent.

Furthermore, the rank adjustment of different convolutional blocks in this work is applied uniformly throughout the U-Net backbone, even though different blocks may have varying highest possible ranks. Future research could explore more adaptive and intelligent strategies for adjusting trainable ranks to enhance training efficiency and performance, as well as identify suitable decay functions and terminal ranks for scenarios beyond those covered in this work.

B. Implications

As discussed in §III, prior to the era of large models, innovations in robot learning primarily focused on learning processes with interaction. However, the increasing size of models has resulted in significantly longer training times, making many previous innovations in online interactive learning less practical due to the time required for policy updates. While the machine learning community has made progress in leveraging the intrinsic rank of large models to improve training efficiency, most of these methods are tailored for fine-tuning rather than training from scratch. This distinction arises from the availability of foundation models in general machine learning, whereas robotics often requires training policies from scratch to address scenario-specific tasks.

Despite years of research in the machine learning community, the concepts of overparameterization and intrinsic ranks remain relatively underexplored in robotics. This work introduces reduced-rank training as a means to address the challenges of training efficiency, thereby making online interactive learning methods more feasible in the era of large models for robot learning. By bridging these gaps, we aim to raise awareness within the robotics community about leveraging overparameterization and intrinsic ranks to design more efficient learning methods while preserving the powerful representations afforded by overparameterized models.

ACKNOWLEDGMENTS

This work was supported by Office of Naval Research award N00014-24-1-2124

REFERENCES

[1] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning. *arXiv preprint arXiv:2012.13255*, 2020.

[2] Peide Cai, Yuxiang Sun, Yuying Chen, and Ming Liu. Vision-based trajectory planning via imitation learning for autonomous vehicles. In *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pages 2736–2742. IEEE, 2019.

[3] Cheng Chi, Zhenjia Xu, Siyuan Feng, Eric Cousineau, Yilun Du, Benjamin Burchfiel, Russ Tedrake, and Shuran Song. Diffusion policy: Visuomotor policy learning via action diffusion. *The International Journal of Robotics Research*, page 02783649241273668, 2023.

[4] Kevin Clark, Paul Vicol, Kevin Swersky, and David J Fleet. Directly fine-tuning diffusion models on differentiable rewards. *arXiv preprint arXiv:2309.17400*, 2023.

[5] Yehuda Dar, Vidya Muthukumar, and Richard G Baraniuk. A farewell to the bias-variance tradeoff? an overview of the theory of overparameterized machine learning. *arXiv preprint arXiv:2109.02355*, 2021.

[6] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: efficient finetuning of quantized llms (2023). *arXiv preprint arXiv:2305.14314*, 52: 3982–3992, 2023.

[7] Simon Du and Jason Lee. On the power of over-parametrization in neural networks with quadratic activation. In *International conference on machine learning*, pages 1329–1338. PMLR, 2018.

[8] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[9] Minho Heo, Youngwoon Lee, Doohyun Lee, and Joseph J Lim. Furniturebench: Reproducible real-world benchmark for long-horizon complex manipulation. *arXiv preprint arXiv:2305.12821*, 2023.

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

[11] Ryan Hoque, Ashwin Balakrishna, Ellen Novoseller, Albert Wilcox, Daniel S Brown, and Ken Goldberg. Thriftydagger: Budget-aware novelty and risk gating for interactive imitation learning. *arXiv preprint arXiv:2109.08273*, 2021.

[12] Ryan Hoque, Ashwin Balakrishna, Carl Putterman, Michael Luo, Daniel S Brown, Daniel Seita, Brijen Thananjeyan, Ellen Novoseller, and Ken Goldberg. Lazydagger: Reducing context switching in interactive imitation learning. In *2021 IEEE 17th international conference on automation science and engineering (case)*, pages 502–509. IEEE, 2021.

[13] Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

[14] Jie Huang, Wei Ge, Hualong Cheng, Chun Xi, Jun Zhu, Fei Zhang, and Weiwei Shang. Real-time obstacle avoidance in robotic manipulation using imitation learning. In *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 976–981. IEEE, 2020.

[15] Jun Jin, Laura Petrich, Masood Dehghan, and Martin Jagersand. A geometric perspective on visual imitation learning. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5194–5200. IEEE, 2020.

[16] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[17] Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Hg-dagger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8077–8083. IEEE, 2019.

[18] Michael Kelly, Chelsea Sidrane, Katherine Driggs-Campbell, and Mykel J Kochenderfer. Hg-dagger: Interactive imitation learning with human experts. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8077–8083. IEEE, 2019.

[19] Diederik P Kingma. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[20] Othmar Koch and Christian Lubich. Dynamical low-rank approximation. *SIAM Journal on Matrix Analysis and Applications*, 29(2):434–454, 2007.

[21] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[22] Alex X Lee, Coline Manon Devin, Yuxiang Zhou, Thomas Lampe, Konstantinos Bousmalis, Jost Tobias Springenberg, Arunkumar Byravan, Abbas Abdolmaleki, Nimrod Gileadi, David Khosid, et al. Beyond pick-and-place: Tackling robotic stacking of diverse shapes. In *5th Annual Conference on Robot Learning*, 2021.

[23] Sung-Wook Lee and Yen-Ling Kuo. Diff-dagger: Uncertainty estimation with diffusion policy for robotic manipulation. *arXiv preprint arXiv:2410.14868*, 2024.

[24] Chunyuan Li, Heerad Farkhoor, Rosanne Liu, and Jason Yosinski. Measuring the intrinsic dimension of objective landscapes. *arXiv preprint arXiv:1804.08838*, 2018.

[25] Yuanzhi Li, Yingyu Liang, and Andrej Risteski. Recovery guarantee of weighted low-rank approximation via

alternating minimization. In *International Conference on Machine Learning*, pages 2358–2367. PMLR, 2016.

[26] Yuanzhi Li, Tengyu Ma, and Hongyang Zhang. Algorithmic regularization in over-parameterized matrix sensing and neural networks with quadratic activations. In *Conference On Learning Theory*, pages 2–47. PMLR, 2018.

[27] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*, 2024.

[28] Antonio Loquercio, Elia Kaufmann, René Ranftl, Matthias Müller, Vladlen Koltun, and Davide Scaramuzza. Learning high-speed flight in the wild. *Science Robotics*, 6(59):eabg5810, 2021.

[29] Ajay Mandlekar, Danfei Xu, Josiah Wong, Soroush Nasiriany, Chen Wang, Rohun Kulkarni, Li Fei-Fei, Silvio Savarese, Yuke Zhu, and Roberto Martín-Martín. What matters in learning from offline human demonstrations for robot manipulation. In *5th Annual Conference on Robot Learning*, 2021.

[30] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[31] Aaditya Prasad, Kevin Lin, Jimmy Wu, Linqi Zhou, and Jeannette Bohg. Consistency policy: Accelerated visuomotor policies via consistency distillation. *arXiv preprint arXiv:2405.07503*, 2024.

[32] 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.

[33] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

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

[35] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[36] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[37] Jonathan Spencer, Sanjiban Choudhury, Arun Venkatraman, Brian Ziebart, and J Andrew Bagnell. Feedback in imitation learning: The three regimes of covariate shift. *arXiv preprint arXiv:2102.02872*, 2021.

[38] Ajay Sridhar, Dhruv Shah, Catherine Glossop, and Sergey Levine. Nomad: Goal masked diffusion policies for navigation and exploration. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 63–70. IEEE, 2024.

[39] Gilbert Strang. *Introduction to linear algebra.* SIAM, 2022.

[40] Xiatao Sun, Shuo Yang, Mingyan Zhou, Kunpeng Liu, and Rahul Mangharam. Mega-dagger: Imitation learning with multiple imperfect experts. *arXiv preprint arXiv:2303.00638*, 2023.

[41] Xiatao Sun, Mingyan Zhou, Zhijun Zhuang, Shuo Yang, Johannes Betz, and Rahul Mangharam. A benchmark comparison of imitation learning-based control policies for autonomous racing. In *2023 IEEE Intelligent Vehicles Symposium (IV)*, pages 1–5. IEEE, 2023.

[42] Xiatao Sun, Francis Fan, Yinxing Chen, and Daniel Rakita. A comparative study on state-action spaces for learning viewpoint selection and manipulation with diffusion policy. *arXiv preprint arXiv:2409.14615*, 2024.

[43] Mojtaba Valipour, Mehdi Rezagholizadeh, Ivan Kobyzev, and Ali Ghodsi. Dylora: Parameter efficient tuning of pre-trained models using dynamic search-free low-rank adaptation. *arXiv preprint arXiv:2210.07558*, 2022.

[44] A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.

[45] Zhendong Wang, Zhaoshuo Li, Ajay Mandlekar, Zhenjia Xu, Jiaojiao Fan, Yashraj Narang, Linxi Fan, Yuke Zhu, Yogesh Balaji, Mingyuan Zhou, et al. One-step diffusion policy: Fast visuomotor policies via diffusion distillation. *arXiv preprint arXiv:2410.21257*, 2024.

[46] Yuwei Wu, Xiatao Sun, Igor Spasojevic, and Vijay Kumar. Deep learning for optimization of trajectories for quadrotors. *IEEE Robotics and Automation Letters*, 2024.

[47] Yuhui Xu, Yuxi Li, Shuai Zhang, Wei Wen, Botao Wang, Wenrui Dai, Yingyong Qi, Yiran Chen, Weiyao Lin, and Hongkai Xiong. Trained rank pruning for efficient deep neural networks. In *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing-NeurIPS Edition (EMC2-NIPS)*, pages 14–17. IEEE, 2019.

[48] Maryam Zare, Parham M Kebria, Abbas Khosravi, and Saeid Nahavandi. A survey of imitation learning: Algorithms, recent developments, and challenges. *IEEE Transactions on Cybernetics*, 2024.

[49] Maryam Zare, Parham M. Kebria, Abbas Khosravi, and Saeid Nahavandi. A survey of imitation learning: Algorithms, recent developments, and challenges. *IEEE Transactions on Cybernetics*, pages 1–14, 2024. doi: 10.1109/TCYB.2024.3395626.

[50] Yanjie Ze, Gu Zhang, Kangning Zhang, Chenyuan Hu, Muhan Wang, and Huazhe Xu. 3d diffusion policy: Generalizable visuomotor policy learning via simple 3d representations. In *ICRA 2024 Workshop on 3D Visual*

Representations for Robot Manipulation, 2024.

[51] Huijie Zhang, Yifu Lu, Ismail Alkhouri, Saiprasad Ravishankar, Dogyoon Song, and Qing Qu. Improving training efficiency of diffusion models via multi-stage framework and tailored multi-decoder architecture. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7372–7381, June 2024.

[52] Xiaoyu Zhang, Matthew Chang, Pranav Kumar, and Saurabh Gupta. Diffusion meets dagger: Supercharging eye-in-hand imitation learning. *arXiv preprint arXiv:2402.17768*, 2024.

[53] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[54] Mingyan Zhou, Biao Wang, Tian Tan, and Xiatao Sun. Developing path planning with behavioral cloning and proximal policy optimization for path-tracking and static obstacle nudging. *arXiv preprint arXiv:2409.05289*, 2024.

[55] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, and Yifeng Zhu. robosuite: A modular simulation framework and benchmark for robot learning. *arXiv preprint arXiv:2009.12293*, 2020.

IX. Appendix

A. Additional Baselines and Variants

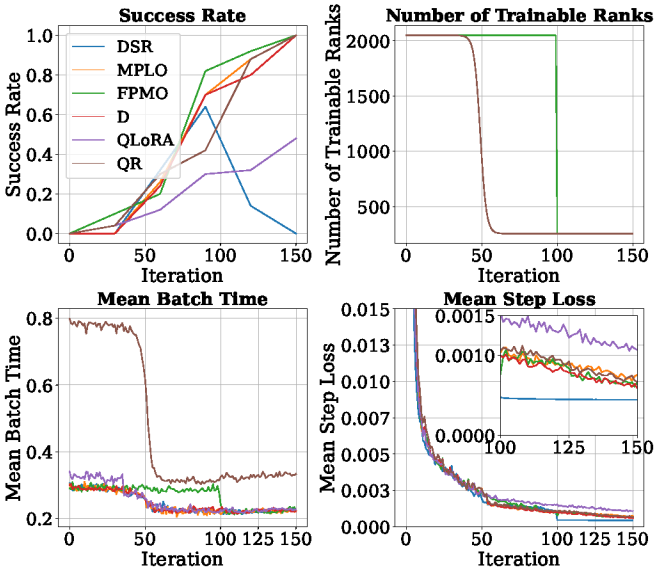

Fig. 9: Results of additional baselines and variants on the MVS Pick and Place task, comparing DRIFT-DAgger (D) to QLoRA (sigmoid decay, steepness 0.5), and four variants: QR (QR decomposition per step), FPMO (full-rank offline, static RM with $r_{min} = 256$ online), MPLO (RM offline, LoRA online), and DSR (fine-tuning last 10% denoising steps online).

TABLE VII: Summary of experiments for additional baseline and variants on the PnP task.

Method	Success Rate	MBT (Offline)	MBT (Online)	MBT (All Stages)
D	**1.0**	**0.26**	**0.22**	**0.24**
QR	**1.0**	0.55	0.33	0.47
FPMO	**1.0**	0.28	0.22	0.26
MPLO	**1.0**	0.26	0.22	0.24
DSR	0.0	0.26	0.22	0.24
QLoRA	0.48	0.27	0.23	0.25

In addition to the baselines considered in the paper, we also compare DRIFT-DAgger (D) with QLoRA [6], and four additional DRIFT variants: QR (using QR decomposition per gradient step, FPMO (full-rank offline, static RM with r_{min} = 256 online), MPLO (RM offline, LoRA online), and DSR (fine-tuning last 10% denoising steps online). The results on the MVS Pick and Place (PnP) task can be found in Table VII and Fig. 9.

First, we observe that QLoRA on the MVS PnP task shows inferior performance to ours. Also, DIRFT-DAgger with QR decomposition reduces training efficiency significantly with negligible performance gains. MPLO perform similarly to DRIFT-DAgger with RM, further suggesting DRIFT outperforms LoRA-based methods due to instability of LoRA from injection and reinstantiation, and its need for well pre-trained weights, whereas DRIFT suits robot learning trained from scratch. FPMO demonstrates slightly worse training efficiency due to full-rank offline training.

DSR causes model collapse during online adaptation despite having lower loss due to overfitting the last few steps during denoising. This is likely because it also originates from diffusion models in image generation [4], which often assume well pre-trained models that are unavailable in diffusion policy context for robot learning.

B. Additional Experiments on the Choice of r_{min} in the Microwave task

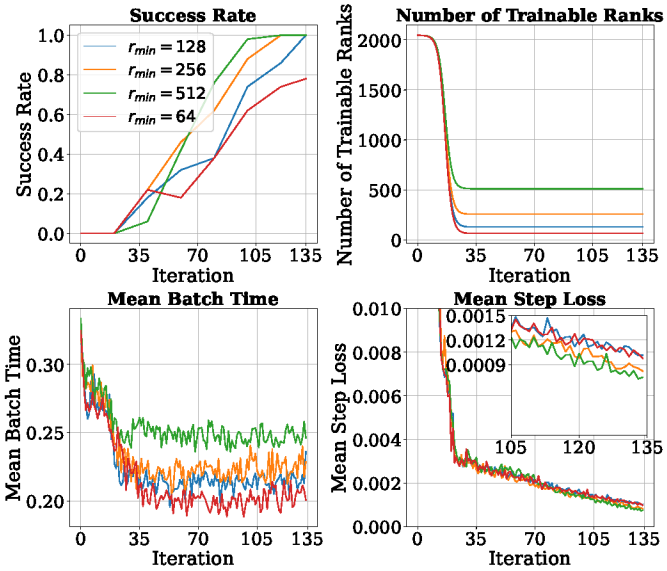

Fig. 10: r_{min} sweep on Microwave task. This task uses 100 offline rollouts, 35 offline epochs, and 100 online iterations to train, whereas the PnP task uses 100 offline rollouts, 100 offline epochs, and 50 online iterations for r_{min} ablation.

TABLE VIII: SSummary of r_{min} sweep results on Microwave task.

r_{min}	Success Rate	MBT (Offline)	MBT (Online)	MBT (All Stages)
64	0.78	**0.25**	**0.20**	**0.21**
128	**1.0**	**0.25**	0.21	0.22
256	**1.0**	0.26	0.22	0.23
512	**1.0**	0.27	0.25	0.25

We also conducted an r_{min} sweep on the Microwave task, complementing the PnP task sweep in the paper. Fig. 10 and Table VIII show that, despite differing tasks and training configuration, the trend is similar to the PnP r_{min} ablation: $r_{min} = 256$ converges slightly faster than $r_{min} = 128$. This suggests $r_{min} = 256$ as a reliable heuristic for manipulation tasks with the specification of diffusion policies per [3].