University of California
Santa Barbara

Towards Generalist Agents through Scaling Offline Reinforcement Learning

A thesis submitted in partial satisfaction
of the requirements for the degree

Masters of Science
in
Computer Science

by

Edwin Tyler Zhang

Committee in charge:

Professor William Wang, Chair
Professor Xifeng Yan
Professor Tobias Höllerer

June 2023

The Thesis of Edwin Tyler Zhang is approved.

Professor Xifeng Yan

Professor Tobias Höllerer

Professor William Wang, Committee Chair

June 2023

Towards Generalist Agents through Scaling Offline Reinforcement Learning

by

Edwin Tyler Zhang

To Ian

Acknowledgements

In no particular order, thank you eternally to Jiachen, William, Amy, Chuang, Yikang, Mengdi, Shun, Ming, Yujie, Jerry B., Eric, Yilun, Michael J, Michael B., Tobias, Xifeng, Ashley, Oier, Lukas, Peter H, Jason, Stephen Z., Stephen R., Andreea, Jason, Michael S., Sharon, Chad, Ellen, Peter V., Jerry G., and my parents.

Abstract

Towards Generalist Agents through Scaling Offline Reinforcement Learning

by

Edwin Tyler Zhang

In recent years, there has been an increasing emphasis on developing generalist agents capable of solving a diverse variety of tasks effectively. We hope that such an agent would be capable of chaining several smaller tasks together, navigating from high-dimensional inputs, and being simple and reliable to train, we will first introduce the current landscape of generalist agents and the state of Deep Reinforcement Learning (RL) and Offline RL. We'll also discuss several major issues underlying these fields, such as training instability and generalization failure. Next, we will explore several proposals for solving such problems, namely through optimization techniques and diffusion. Finally, we will discuss the major challenges and opportunities that lie ahead in the future for training generalist agents.

Contents

Abstract **vi**

1 Introduction **1**
1.1 Solving the hardest problem 1
1.2 Learnings from NLP 3
1.3 Offline RL 4
1.4 Contribution 5

2 Background **8**
2.1 Reinforcement Learning. 8
2.2 Behavior Constrained Policy Optimization. 9
2.3 Language Conditioned RL. 9
2.4 Goal Conditioned Imitation Learning and Hierarchical RL. 10

3 Closed-form Policy Improvement **12**
3.1 Introduction 12
3.2 Closed-Form Policy Improvement 14
3.3 Related Work 22
3.4 Experiments 26
3.5 Proofs and Theoretical Results 34
3.6 Detailed Procedures to obtain Equation 3.13 48
3.7 Multi-step and iterative algorithms 51
3.8 CFPI beyond Gaussian policies 53
3.9 Reliable evaluation to address the statistical uncertainty 57
3.10 Hyper-parameter settings and training details 58
3.11 Additional Experiments 66
3.12 Conclusion and Limitations 77

4 Scaling Diffusion for Offline RL **78**
4.1 Introduction 78
4.2 The Language Control Diffusion (LCD) Framework 82

4.3 Experiments . . . 87
4.4 Related Work . . . 94
4.5 Hyper-parameter settings and training details . . . 95
4.6 Training Objective Derivation . . . 97
4.7 Proof of 4.2.1 . . . 100
4.8 Diffuser-2D . . . 102
4.9 Task Distribution . . . 103
4.10 Representation Failures . . . 104
4.11 TSNE Comparison between Groud Truth (GT) trajectory and Diffuser-1D (DM) trajectory . . . 106
4.12 HULC Latent Plan TSNE . . . 108
4.13 Comparison of success rates (SR) across single tasks, evaluated with HULC and LCD . . . 109
4.14 Model Card for Language Control Diffusion . . . 110
4.15 Conclusion and Limitations . . . 112

5 Conclusion **113**
5.1 Contribution . . . 113
5.2 Limitations . . . 113

A Appendix **115**
A.1 Table of Abbreviations . . . 115
A.2 List of Figures . . . 117
A.3 List of Tables . . . 122
A.4 List of Algorithms . . . 126

Bibliography **127**

Chapter 1

Introduction

1.1 Solving the hardest problem

The pursuit of Artificial General Intelligence (AGI) has been driven by numerous motivations, each distinct yet intertwined in their objective to further the boundaries of the intellectual capabilities of humanity. One particularly compelling reason for pursuing AGI originates from a perspective seemingly separate from the domain of AI itself - the human yearning to improve our world, to unravel and conquer some of the most formidable challenges encounterable to society. Some examples may include climate change, war, economic inequality, cancer, corruption, sex-trafficking, and homelessness.

Engaging in the quest to solve the hardest possible social problems serves as a testament to the relentless human spirit that ceaselessly aspires for advancement and achievement. The paradox, however, lies in the realization that our own intelligence and computational prowess, although remarkable, are inherently limited. Acknowledging this fundamental constraint kindles the logical progression towards building a system that surpasses our own intellectual abilities. Therefore, the fundamental question that drives this thesis is this: 'How does one draw closer towards the goal of building an agent more intelligent

than ourselves to improve our world?'

Before delving into the mechanics of building such a system, one must pause to define 'intelligence'. Here, intelligence is defined as the capacity to accomplish a set objective, with the level of intelligence commensurate to the difficulty of the goal. Under this definition, Reinforcement Learning (RL) along with Markov Decision Processes (MDPs) offer a robust, general learning framework for characterizing intelligence [1]. This definition elegantly aligns with how computational problems are depicted - as functions that map instances to solutions. Such problems can be reformulated as Reinforcement Learning (RL) problems, where a reward of 1 is assigned for a correct solution, and 0 otherwise.

Here, I introduce the 'Computability Hypothesis', which asserts that almost all problems of practical significance are computable. Thus this hypothesis implies that nearly all practical problems can also be represented as a RL problem. Our paradigm equates the two. Solving RL is solving intelligence. Therefore, creating practical AGI can be seen as solving the hardest RL problem.

A natural question then arises from this reasoning: How does one gauge the 'hardness' of an RL problem? To answer this, I propose adopting a paradigm akin to computational complexity theory. Imagine the optimal RL algorithm for a given problem - the 'hardness' of the problem could then be determined by the time and memory resources required to run this algorithm. Thus, characterizing the hardness of MDPs is essential. Several features contribute to this hardness measure for a given Markov Decision Process, including the size of the state space, the size of individual states, and the sparsity of the reward function. Other features worthy of note including the branching factor, horizon length, credit assignment, spurious correlation, environment complexity (transition function complexity), uncertainty, partial observability, nonhomogeneity, nonergodicity, limited state space coverage, and exploration. While these are properties identifiable to the research community now, numerous other undiscovered factors may also play significant

roles. It is left to future work to formulate a quantitative 'hardness' measure from these properties. Here, I proceed to the next section, where we begin to try to characterize the properties of successful learning algorithms from Natural Language Processing (NLP), and apply these lessons to RL.

1.2 Learnings from NLP

Drawing from the realm of Natural Language Processing (NLP), which has seen tremendous successes in recent years, one can find potential avenues to address some of these challenges. Applying lessons from NLP to RL can provide valuable insights and innovative strategies to model and solve complex problems. This thesis, therefore, embarks on the ambitious journey of deciphering properties of AGI, drawing from the recent success of NLP.

The field of Natural Language Processing (NLP) offers three crucial insights that significantly guide our approach towards AGI: firstly, scaling the model, or amplifying the amount of computation; secondly, augmenting the quantity of data at our disposal [2]; and thirdly, the usage of the transformer architecture [3].

The act of scaling the model is a critical factor that cannot be overstated. Nonetheless, it must be emphasized that this scaling attains its true potential when performed in conjunction with the expansion of data. However, simply increasing the model size without feeding it with enough data to learn from can also lead to poor performance. The inverse also holds - even with an abundance of data, if the model capacity is insufficient, the full value of the data cannot be leveraged. Thus, the concurrent growth of both model size and data quantity forms a symbiotic relationship that fuels the overall learning capacity of the system.

Further, the transformer architecture plays a pivotal role in this pursuit. This

groundbreaking architecture enables the parallelization of sequential processing, thereby providing the necessary bandwidth to accommodate scaling. Beyond the benefits of computational efficiency and scalability, transformers also give rise to emergent properties such as in-context learning. These structures learn from their input data in a dynamic way, adapting their output based on the input they're currently processing. This capability opens up a myriad of possibilities in tasks that require understanding the context, making transformers a compelling choice for advanced learning systems.

Building on these insights, our goal then is to adapt and apply these lessons to the field of Reinforcement Learning (RL). Can one harness the transformative potential of these elements from NLP, and infuse it within RL algorithms to pave the way for more advanced, and potentially general, forms of artificial intelligence? By exploring this question, one embarks on a journey towards the realization of AGI, guided by the lessons of NLP and the potential of RL.

1.3 Offline RL

In this thesis, the concentration will be on a specific subset of Reinforcement Learning called Offline Reinforcement Learning (Offline RL), also referred to as batch RL [4]. Offline RL provides an avenue to learn policies directly from a fixed batch of data or a dataset without further interactions with the environment. The key challenge in Offline RL is to maximize reward and to derive optimal policies based on the existing data, rather than exploring the environment for new data, as is done in traditional online RL.

The practical implications of this approach are quite profound. Offline RL presents an attractive option for scenarios where online RL is either impractical or unsafe. This often occurs in real-world problems where a simulator might not exist or where real-time evaluation of the environment is costly, risky, or outright dangerous. For instance, in

the context of self-driving cars, it would be reckless to allow a partially trained model to operate in a live environment, potentially risking lives and property. Offline RL offers a more viable solution in such cases, where learning can be accomplished using historical driving data, mitigating these risks.

The inherent nature of Offline RL also makes it a far more scalable problem than its online counterpart. By relying on fixed datasets, Offline RL eliminates the need for costly, continuous environmental interactions and allows for greater computational parallelization, enhancing scalability. This framework is greatly inspired by the success of NLP, where the availability of enormous datasets and the capacity to train on these datasets offline has revolutionized the field. By aligning Offline RL with the principles that have led to breakthroughs in NLP, new pathways can be potentially unlocked towards AGI, leveraging the benefits of both fields to drive our understanding and development of intelligence beyond our current limitations.

1.4 Contribution

In alignment with this overarching vision, the first contribution in this thesis is the development of a technique designed to bypass the often volatile Offline RL algorithms by using only stable, supervised learning algorithms to allow future scaling of the model and data.

This method, "Closed-form Policy Improvement" [5], builds upon a unique mathematical approach. At the heart of it lies a Taylor Approximation applied to the estimate of the value of an action in a given state. This estimate, widely known in the field of RL as the Q-function, provides a measure of how beneficial a particular action is in a given state. The Closed-form Policy Improvement method leverages this Q-function approximation to improve the policy directly, given a Gaussian probability distribution over actions. This

Gaussian distribution represents the uncertainty about the best action to take at a given time. By taking a Taylor approximation of the Q-function, one can simplify the Policy Improvement step to a Quadratically Constrained Linear Program (QCLP), allowing us to solve Policy Improvement in closed-form.

This direct, closed-form solution offers a distinct advantage over traditional methods, enabling the algorithm to train more stably and efficiently. By bringing together aspects of supervised learning and RL in this unique way, the Closed-form Policy Improvement method offers a powerful approach to handle the challenges associated with Offline RL, paving the way for greater scalability and practical applicability in real-world generalist agent implementations.

Following this initial breakthrough, the second major contribution follows: a method for scaling Offline RL through the adoption of diffusion. The term 'diffusion' in this context refers to a recently successful paradigm for generative modeling, which has gained prominence through its application in models like Stable Diffusion and DALLE-2 [6, 7]. These models, while highly effective, are inherently computationally intensive as they require multiple forward passes to generate outputs, a factor that can limit their practical utility.

To address these challenges, the concept of a hierarchical policy in the diffusion process is introduced [8], offering a novel approach to mitigating these computational demands. The hierarchical policy essentially breaks down the problem into manageable, hierarchical subtasks, making it possible to compute more efficiently and thereby offsetting the inherent expense of the diffusion approach.

This introduction of a hierarchical policy facilitates an expansion in the reach of Offline RL, enabling its application to a broader range of tasks, including longer tasks and those involving higher-dimensional state spaces such as images. By effectively surmounting the computational hurdles associated with diffusion-based modeling, one opens up new

horizons for Offline RL, significantly enhancing its potential for scalability and applicability to a wide range of complex, real-world problems.

Chapter 2

Background

2.1 Reinforcement Learning.

RL aims to maximize returns in a Markov Decision Process (MDP) [1] $\mathcal{M} = (\mathcal{S}, \mathcal{A}, R, T, \rho_0, \gamma)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, reward function R, transition function T, initial state distribution ρ_0, and discount factor $\gamma \in [0, 1)$. At each time step t, the agent starts from a state $s_t \in \mathcal{S}$, selects an action $a_t \sim \pi(\cdot|s_t)$ from its policy π, transitions to a new state $s_{t+1} \sim T(\cdot|s_t, a_t)$, and receives reward $r_t := R(s_t, a_t)$. We define the action value function associated with π by $Q^\pi(s, a) = \mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, a_0 = a]$. The goal of an RL agent is to learn an optimal policy π^* that maximizes the expected discounted cumulative reward without access to the ground truth R and T and can thus be formulated as

$$\pi^* = \arg\max_\pi J(\pi) := \mathbb{E}_{s\sim\rho_0, a\sim\pi(\cdot|s)}[Q^\pi(s, a)] \tag{2.1}$$

In this thesis, we consider *offline* RL settings, where we assume restricted access to the MDP $\mathcal{M}$, and a previously collected dataset $\mathcal{D}$ with N transition tuples $\{(s_t^i, a_t^i, r_t^i)\}_{i=1}^N$. We denote the underlying policy that generates $\mathcal{D}$ as π_β, which may or may not be a

mixture of individual policies.

2.2 Behavior Constrained Policy Optimization.

One of the critical challenges in offline RL is that the learned Q function tends to assign spuriously high values to OOD actions due to extrapolation error, which is well documented in previous literature [4, 9]. Behavior Constrained Policy Optimization (BCPO) methods [4, 9, 10, 11, 12] explicitly constrain the action selection of the learned policy to stay close to the behavior policy π_β, resulting in a policy improvement step that can be generally summarized by the optimization problem below:

$$\max_{\pi} \mathbb{E}_{s\sim\mathcal{D}} \left[\mathbb{E}_{\tilde{a}\sim\pi(\cdot|s)} \left[Q\left(s,\tilde{a}\right)\right] - \alpha D\left(\pi(\cdot \mid s), \pi_\beta(\cdot \mid s)\right)\right], \tag{2.2}$$

where $D(\cdot,\cdot)$ is a divergence function that calculates the divergence between two action distributions, and α is a hyper-parameter controlling the strength of regularization. Consequently, the policy is optimized to maximize the Q-value while staying close to the behavior distribution.

Different algorithms may choose different $D(\cdot,\cdot)$ (e.g., KL Divergence [11, 13], Rényi divergence [14, 15], MSE [10] and MMD [9]).

2.3 Language Conditioned RL.

We additionally consider a language-conditioned RL setting where we assume that the true reward function $\mathcal{R}$ is unknown, and must be inferred from a natural language instruction $L \in \mathcal{L}$. Formally, let $\mathcal{F}$ be the function space of $\mathcal{R}$. Then the goal becomes learning an operator from the language instruction to a reward function $\psi : \mathcal{L} \mapsto \mathcal{F}$,

and maximizing the policy objective conditioned on the reward function $\psi(L)$: $J(\pi(\cdot \mid s, \mathcal{R})) = \mathbb{E}_{a\sim\pi, s\sim p} \sum_{t=0}^{\infty} \gamma^t r_t$. This formulation can be seen as a contextual MDP [16], where language is seen as a context variable that affects reward but not dynamics. Note that the space of tasks that can be specified by language is much larger than that of reward, due to the Markov restriction of the latter [17]. For example, "pour the milk" and "pour the milk after five o'clock" are both valid instructions, but are indistinguishable from a reward function if the state does not contain temporal information. We assume access to a prior collected dataset $\mathcal{D}$ of N annotated trajectories $\tau_i = \langle (s_0, a_0, ...s_T), L_i \rangle$. It will be clear from the context whether we are considering a regular dataset introduced in the prior section or a language-conditioned dataset. The language conditioned policy π_β, or the behavior policy, is defined to be the policy that generates the aforementioned dataset. This general setting can be further restricted to prohibit environment interaction, which recovers offline RL or imitation learning (IL). In this paper, we assume access to a dataset of expert trajectories, such that π_β = optimal policy $\pi^\star$. Although this may seem like a strong assumption, the setting still poses a challenging learning problem as many unseen states will be encountered during evaluation and must be generalized to. Several prior methods have failed in this setting [18, 19].

2.4 Goal Conditioned Imitation Learning and Hierarchical RL.

Goal-conditioned reinforcement learning (RL) is a subfield of RL that focuses on learning policies that can achieve specific goals or objectives, rather than simply maximizing the cumulative reward. This approach has been extensively studied in various forms in the literature [20, 21, 22, 23, 24], although we focus on the hierarchical approach [25].

Following [26], we formulate the framework in a two-level manner where a high level policy $\pi_{\text{hi}}(\tilde{a}|s)$ samples a goal state $g = \tilde{a}$ every c time steps. We refer to c as the temporal stride. A low level policy (LLP) $\pi_{\text{lo}}(a|s_t, g_t)$ then attempts to reach this state, which can be trained through hindsight relabelling [27] the language-conditioned offline dataset $\mathcal{D}$.

Chapter 3

Closed-form Policy Improvement

3.1 Introduction

The deployment of Reinforcement Learning (RL) [1] in real-world applications is often hindered by the large amount of online data it requires. Implementing an untested policy can be costly and dangerous in fields such as robotics [28] and autonomous driving [29]. To address this issue, offline RL (a.k.a batch RL) [30, 31] has been proposed to learn a policy directly from historical data without environment interaction. However, learning competent policies from a static dataset is challenging. Previous research has shown that learning a policy without constraining its deviation from the data-generating policies suffers from significant extrapolation errors, leading to training divergence [4, 9].

Current literature has demonstrated two successful paradigms for managing the trade-off between policy improvement and limiting the distributional shift from the behavior policies. Under the actor-critic framework [32], behavior constrained policy optimization (BCPO) [4, 9, 10, 11, 12, 33, 34, 35, 36] explicitly regularizes the divergence between learned and behavior policies, while conservative methods [37, 38, 39, 40, 41, 42] penalize the value estimate for out-of-distribution (OOD) actions to avoid overestimation errors.

However, most existing model-free offline RL algorithms use stochastic gradient descent (SGD) to optimize their policies, which can lead to instability during the training process and require careful tuning of the number of gradient steps. As highlighted by [10], the performance of the offline-trained policies can be influenced by the specific stopping point chosen for evaluation, with substantial variations often observed near the final stage of training. This instability poses a significant challenge in offline RL, given the restricted access to environment interaction makes it difficult to perform hyper-parameter tuning. In addition to the variations across different stopping points, our experiment in Table 3.4 reveals that using SGD for policy improvement can result in significant performance variations across different random seeds, a phenomenon well-documented for online RL algorithms as well [43].

In this chapter, we aim to mitigate the aforementioned learning instabilities of offline RL by designing stable policy improvement operators. In particular, we take a closer look at the BCPO paradigm and make a novel observation that the requirement of limited distributional shift motivates the use of the first-order Taylor approximation [44], leading to a linear approximation of the policy objective that is accurate in a sufficiently small neighborhood of the behavior action. Based on this crucial insight, we construct our policy improvement operators that return closed-form solutions by carefully designing a tractable behavior constraint. When modeling the behavior policies as a Single Gaussian, our policy improvement operator deterministically shifts the behavior policy towards a value-improving direction derived by solving a Quadratically Constrained Linear Program (QCLP) in closed form. As a result, our method avoids the training instability in policy improvement since it only requires learning the underlying behavior policies of a given dataset, which is a supervised learning problem.

Furthermore, we note that practical datasets are likely to be collected by heterogeneous policies, which may give rise to a multimodal behavior action distribution. In this scenario,

a Single Gaussian will fail to capture the multiple modes of the underlying distribution, limiting the potential for policy improvement. While modeling the behavior as a Gaussian Mixture provides better expressiveness, it incurs extra optimization difficulties due to the non-concavity of its log-likelihood. We tackle this issue by leveraging the LogSumExp's lower bound and Jensen's inequality, again leading to a closed-form policy improvement (CFPI) operator applicable to multimodal behavior policies. Empirically, we demonstrate the effectiveness of Gaussian Mixture over the conventional Single Gaussian when the underlying distribution comes from heterogenous policies.

In summary, our main contributions are threefold:

- CFPI operators that are compatible with single mode and multimodal behavior policies and can be leveraged to improve policies learned by the other algorithms.
- Empirical evidence showing the benefits of modeling the behavior policy as a Gaussian Mixture.
- One-step and iterative instantiations of our algorithm that outperform state-of-the-art (SOTA) algorithms on the standard D4RL benchmark [45].

3.2 Closed-Form Policy Improvement

In this section, we introduce our policy improvement operators that map the behavior policy to a higher-valued policy, which is accomplished by solving a linearly approximated BCPO. We first show that modeling the behavior policy as a Single Gaussian transforms the approximated BCPO into a QCLP and thus can be solved in closed-form (Sec. 3.2.1). Given that practical datasets are usually collected by heterogeneous policies, we generalize the results by modeling the behavior policies as a Gaussian Mixture to facilitate expressiveness and overcome the incurred optimization difficulties by leveraging

the LogSumExp's lower bound (LB) and Jensen's Inequality (Sec. 3.2.2). We close this section by presenting an offline RL paradigm that leverages our policy improvement operators (Sec. 3.2.3).

3.2.1 Approximated behavior constrained optimization

We aim to design a learning-free policy improvement operator to avoid learning instability in offline settings. We observe that optimizing towards BCPO's policy objective (2.2) induces a policy that admits limited deviation from the behavior policy. Consequently, it will only query the Q-value within the neighborhood of the behavior action during training, which naturally motivates the employment of the first-order Taylor approximation to derive the following linear approximation of the Q function

$$\begin{aligned} \bar{Q}(s, a; a_\beta) &= (a - a_\beta)^T \left[\nabla_a Q(s, a)\right]_{a=a_\beta} + Q(s, a_\beta) \\ &= a^T \left[\nabla_a Q(s, a)\right]_{a=a_\beta} + \text{const.} \end{aligned} \tag{3.1}$$

By Taylor's theorem [44], $\bar{Q}(s, a; a_\beta)$ only provides an accurate linear approximation of $Q(s, a)$ in a sufficiently small neighborhood of a_β. Therefore, the choice of a_β is critical.

Recognizing (2.2) as a Lagrangian and with the linear approximation (3.1), we propose to solve the following surrogate problem of (2.2) given any state s:

$$\begin{aligned} \max_{\pi} \quad & \mathop{\mathbb{E}}_{\tilde{a}\sim\pi} \left[\tilde{a}^T \left[\nabla_a Q(s, a)\right]_{a=a_\beta}\right], \\ \text{s.t.} \quad & D\left(\pi(\cdot \mid s), \pi_\beta(\cdot \mid s)\right) \leq \delta. \end{aligned} \tag{3.2}$$

Note that it is not necessary for $D(\cdot, \cdot)$ to be a (mathematically defined) divergence measure since any generic $D(\cdot, \cdot)$ that can constrain the deviation of π's action from π_β can be considered.

Single Gaussian Behavior Policy. In general, (3.2) does not always have a closed-

form solution. We analyze a special case where $\pi_\beta = \mathcal{N}(\mu_\beta, \Sigma_\beta)$ is a Gaussian policy, $\pi = \mu$ is a deterministic policy, and $D(\cdot, \cdot)$ is a negative log-likelihood function. In this scenario, a reasonable choice of μ should concentrate around μ_β to limit distributional shift. Therefore, we set $a_\beta = \mu_\beta$ and the optimization problem (3.2) becomes the following:

$$\max_{\mu} \ \mu^T \left[\nabla_a Q(s,a)\right]_{a=\mu_\beta}, \ \text{s.t.} \ -\log \pi_\beta(\mu|s) \leq \delta \tag{3.3}$$

We now show that (3.3) has a closed-form solution.

Proposition 3.2.1. The optimization problem (3.3) has a closed-form solution that is given by

$$\mu_{\text{sg}}(\tau) = \mu_\beta + \frac{\sqrt{2\log\tau}\,\Sigma_\beta \left[\nabla_a Q(s,a)\right]_{a=\mu_\beta}}{\left\| \left[\nabla_a Q(s,a)\right]_{a=\mu_\beta} \right\|_{\Sigma_\beta}}, \tag{3.4}$$

where $\delta = \frac{1}{2}\log\det(2\pi\Sigma_\beta) + \log\tau$ and $\|x\|_\Sigma = \sqrt{x^T\Sigma x}$.

Proof sketch. (3.3) can be converted into the QCLP below that has a closed-form solution given by (3.4).

$$\begin{aligned} \max_{\mu} \quad & \mu^T \left[\nabla_a Q(s,a)\right]_{a=\mu_\beta}, \\ \text{s.t.} \quad & \frac{1}{2}(\mu-\mu_\beta)^T\Sigma_\beta^{-1}(\mu-\mu_\beta) \leq \log\tau \end{aligned} \tag{3.5}$$

A full proof is given in subsection 3.5.1. □

Although we still have to tune τ as tuning α in (2.2) for conventional BCPO methods, we have a transparent interpretation of τ's effect on the action selection thanks to the tractability of (3.3). Due to the KKT conditions [46], the μ_{sg} returned by (3.4) has the following property

$$\begin{aligned} & \log\pi_\beta(\mu_{\text{sg}}|s) = -\delta = \log\frac{\pi_\beta(\mu_\beta|s)}{\tau} \\ \iff \quad & \pi_\beta(\mu_{\text{sg}}|s) = \frac{\pi_\beta(\mu_\beta|s)}{\tau} \end{aligned}. \tag{3.6}$$

While setting $\tau = 1$ will always return the mean of π_β, a large τ might send μ_{sg} out of the support of π_β, breaking the accuracy guarantee of the first-order Taylor approximation.

3.2.2 Gaussian Mixture as a more expressive model

Performing policy improvement with (3.4) enjoys favorable computational efficiency and avoids the potential instability caused by SGD. However, its tractability relies on the Single Gaussian assumption of the behavior policy π_β. In practice, the historical datasets are usually collected by heterogeneous policies with different levels of expertise. A Single Gaussian may fail to capture the whole picture of the underlying distribution, motivating the use of a Gaussian Mixture to represent π_β.

$$\pi_\beta = \sum_{i=1}^{N} \lambda_i \mathcal{N}(\mu_i, \Sigma_i), \quad \sum_{i=1}^{N} \lambda_i = 1 \tag{3.7}$$

However, directly plugging the Gaussian Mixture π_β into (3.3) breaks its tractability, resulting in a non-convex optimization

$$\begin{aligned} \max_{\mu} \quad & \mu^T \left[\nabla_a Q(s,a)\right]_{a=a_\beta}, \\ \text{s.t.} \quad & \log \sum_{i=1}^{N} \Big(\lambda_i \det(2\pi\Sigma_i)^{-\frac{1}{2}} \\ & \qquad \cdot \exp\big(-\frac{1}{2}(\mu-\mu_i)^T \Sigma_i^{-1}(\mu-\mu_i)\big)\Big) \geq -\delta \end{aligned} \tag{3.8}$$

We are confronted with two major challenges to solve the optimization problem (3.8). First, it is unclear how to choose a proper a_β while we need to ensure that the solution μ lies within a small neighborhood of a_β. Second, the constraint of (3.8) does not admit a convex form, posing non-trivial optimization difficulties. We leverage the lemma below to tackle the non-convexity of the constraint.

Lemma 3.2.1. $\log \sum_{i=1}^{N} \lambda_i \exp(x_i)$ *admits the following inequalities:*

1. *(LogSumExp's LB)*

   $\log \sum_{i=1}^{N} \lambda_i \exp(x_i) \geq \max_i \{x_i + \log \lambda_i\}$

2. *(Jensen's Inequality)*

   $\log \sum_{i=1}^{N} \lambda_i \exp(x_i) \geq \sum_{i=1}^{N} \lambda_i x_i$

Next, we show that applying each inequality in Lemma 3.2.1 to the constraint of (3.8) respectively resolves the intractability and leads to natural choices of a_β.

Proposition 3.2.2. By applying the first inequality of Lemma 3.2.1 to the constraint of (3.8), we can derive an optimization problem that lower bounds (3.8)

$$\begin{aligned} \max_{\mu} \quad & \mu^T \left[\nabla_a Q(s,a)\right]_{a=a_\beta}, \\ \text{s.t.} \quad & \max_i \Big\{ -\frac{1}{2}(\mu-\mu_i)^T \Sigma_i^{-1}(\mu-\mu_i) \\ & \qquad -\frac{1}{2}\log\det(2\pi\Sigma_i) + \log\lambda_i \Big\} \geq -\delta, \end{aligned} \tag{3.9}$$

and the closed-form solution to (3.9) is given by

$$\begin{aligned} \mu_{\text{lse}}(\tau) = \arg\max_{\overline{\mu}_i(\delta)} \quad & \overline{\mu}_i{}^T \left[\nabla_a Q(s,a)\right]_{a=\mu_i}, \\ \text{s.t.} \quad & \delta = \min_i \{\frac{1}{2}\log\det(2\pi\Sigma_i) - \log\lambda_i\} + \log\tau, \\ \text{where} \quad & \overline{\mu}_i(\delta) = \mu_i + \frac{\kappa_i \Sigma_i \left[\nabla_a Q(s,a)\right]_{a=\mu_i}}{\left\| \left[\nabla_a Q(s,a)\right]_{a=\mu_i} \right\|_{\Sigma_i}}, \\ \text{and} \quad & \kappa_i = \sqrt{2(\delta + \log\lambda_i) - \log\det(2\pi\Sigma_i)}\ . \end{aligned} \tag{3.10}$$

Proposition 3.2.3. By applying the second inequality of Lemma 3.2.1 to the constraint of

(3.8), we can derive an optimization problem that lower bounds (3.8)

$$\begin{aligned} \max_{\mu} \quad & \mu^T \left[\nabla_a Q(s,a)\right]_{a=a_\beta}, \\ \text{s.t.} \quad & \sum_{i=1}^{N} \lambda_i \Big(-\frac{1}{2}\log\det(2\pi\Sigma_i) \\ & \qquad -\frac{1}{2}(\mu-\mu_i)^T\Sigma_i^{-1}(\mu-\mu_i)\Big) \geq -\delta \end{aligned} \tag{3.11}$$

and the closed-form solution to (3.11) is given by

$$\begin{aligned} \mu_{\text{jensen}}(\tau) &= \overline{\mu} + \frac{\kappa_i \overline{\Sigma}\left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}}}{\left\|\left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}}\right\|_{\overline{\Sigma}}}, \quad \text{where} \\ \kappa_i &= \sqrt{2\log\tau - \sum_{i=1}^{N}\lambda_i\mu_i^T\Sigma_i^{-1}\mu_i + \overline{\mu}^T\overline{\Sigma}^{-1}\overline{\mu}}\,, \\ \overline{\Sigma} &= \left(\sum_{i=1}^{N}\lambda_i\Sigma_i^{-1}\right)^{-1}, \ \overline{\mu} = \overline{\Sigma}\left(\sum_{i=1}^{N}\lambda_i\Sigma_i^{-1}\mu_i\right), \\ \delta &= \log\tau + \frac{1}{2}\sum_{i=1}^{N}\lambda_i\log\det(2\pi\Sigma_i) \end{aligned} \tag{3.12}$$

We defer the detailed proof of Proposition 3.2.2 and Proposition 3.2.3 as well as how we choose a_β for each optimization problem to subsection 3.5.2 and subsection 3.5.3, respectively.

Comparing to the original optimization problem (3.8), both problems (3.9) and (3.11) impose more strict trust region constraints, which is accomplished by enforcing the lower bound of the log probabilities of the Gaussian Mixture to exceed a certain threshold, with τ controlling the size of the trust region. Indeed, these two optimization problems have their own assets and liabilities. When π_β exhibits an obvious multimodality as is shown in Fig. 4.1 (L), the lower bound of $\log\pi_\beta$ constructed by Jensen's Inequality cannot capture different modes due to its concavity, losing the advantage of modeling π_β as a Gaussian

Mixture. In this case, the optimization problem (3.9) can serve as a reasonable surrogate problem of (3.8), as LogSumExp's LB still preserves the multimodality of $\log \pi_\beta$.

When π_β is reduced to a Single Gaussian, the approximation with the Jensen's Inequality becomes equality as is shown in Fig. 4.1 (M). Thus μ_{jensen} returned by (3.12) exactly solves the optimization problem (3.8). However, in this case, the tightness of LogSumExp's LB largely depends on the weights $\lambda_{i=1...N}$. If each Gaussian component is distributed and weighted identically, the lower bound will be $\log N$ lower than the actual value. Moreover, there also exists the scenario (Fig. 4.1 (R)) when both (3.9) and (3.11) can serve as reasonable surrogates to the original problem (3.8).

Fortunately, we can combine the best of both worlds and derives a CFPI operator accounting for all the above scenarios, which returns a policy that selects the higher-valued action from μ_{lse} and μ_{jensen}

$$\mu_{\text{mg}}(\tau) = \underset{\mu \in \{\mu_{\text{lse}}(\tau), \mu_{\text{jensen}}(\tau)\}}{\arg\max} Q(s, \mu) \tag{3.13}$$

3.2.3 Algorithm template

We have derived two CFPI operators that map the behavior policy to a higher-valued policy. When the behavior policy π_β is a Single Gaussian, $\mathcal{I}_{\text{SG}}(\pi_\beta, Q; \tau)$ returns a policy with action selected by (3.4). When π_β is a Gaussian Mixture, $\mathcal{I}_{\text{MG}}(\pi_\beta, Q; \tau)$ returns a policy with action selected by (3.13). We note that our methods can also work with a non-Gaussian π_β. section 3.8 provides the derivations for the corresponding CFPI operators when π_β is modeled as both a deterministic policy and VAE. Algorithm 1 shows that our CFPI operators enable the design of a general offline RL template that can yield one-step, multi-step and iterative methods, where $\mathcal{E}$ is a general policy evaluation operator that returns a value function $\hat{Q}_t$. When setting $T = 0$, we obtain our one-step

Algorithm 1 Offline RL with CFPI operators

Input: Dataset $\mathcal{D}$, baseline policy $\hat{\pi}_b$, value function $\hat{Q}_{-1}$, HP τ

1: Warm start $\hat{Q}_0 = \text{SARSA}(\hat{Q}_{-1}, \mathcal{D})$ with the SARSA-style algorithm [1]
2: Get one-step policy $\hat{\pi}_1 = \mathcal{I}(\hat{\pi}_b, \hat{Q}_0; \tau)$
3: **for** $t = 1 \dots T$ **do**
4: Policy evaluation: $\hat{Q}_t = \mathcal{E}(\hat{Q}_{t-1}, \hat{\pi}_t, \mathcal{D})$
5: Get policy: $\hat{\pi}_{t+1} = \mathcal{I}(\hat{\pi}_b, \hat{Q}_t; \tau)$ (concrete choices of $\mathcal{I}$ includes $\mathcal{I}_{\text{MG}}$ and $\mathcal{I}_{\text{SG}}$)
6: **end for**

method. We defer the discussion on multi-step and iterative methods to the section 3.7.

While the design of our CFPI operators is motivated by the behavior constraint, we highlight that they are compatible with general baseline policies π_b besides π_β. Sec. 3.4.2 and subsection 3.11.7 show that our CFPI operators can improve policies learned by IQL and CQL [37].

3.2.4 Theoretical guarantees for CFPI operators

At a high level, Algorithm 1 follows the *approximate policy iteration* (API) [47] by iterating over the policy evaluation ($\mathcal{E}$ step, Line 4) and policy improvement ($\mathcal{I}$ step, Line 5). Therefore, to verify $\mathcal{E}$ provides the improvement, we need to first show policy evaluation $\hat{Q}_t$ is accurate. We employ the Fitted Q-Iteration [1] to perform policy evaluation, which is known to be statistically efficient (*e.g.* [48]) under the mild condition for the function approximation class. Next, for the performance gap between $J(\hat{\pi}_{t+1}) - J(\hat{\pi}_t)$, we apply the standard performance difference lemma [49, 50].

Theorem 3.2.2. *[Safe Policy Improvement] Assume the state and action spaces are discrete.*[1] *Let $\hat{\pi}_1$ be the policy obtained after the CFPI update (Line 2 of Algorithm 1).*

[1]The assumption of discreteness is made only for the purpose of analysis. For the more general cases, please refer to subsection 3.5.4.

Then with probability $1-\delta$*,*

$$\begin{aligned}
&J(\hat{\pi}_1) - J(\hat{\pi}_\beta) \\
\geq &\frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\hat{\pi}_1}}\left[\bar{Q}^{\hat{\pi}_\beta}(s,\hat{\pi}_1(s)) - \bar{Q}^{\hat{\pi}_\beta}(s,\hat{\pi}_\beta(s))\right] \\
-&\frac{2}{1-\gamma}\mathbb{E}_{s\sim d^{\hat{\pi}_1}}\mathbb{E}_{a\sim\hat{\pi}_1(\cdot|s)}\left[\frac{C_{\gamma,\delta}}{\sqrt{\mathcal{D}(s,a)}} + C_{\text{CFPI}}(s,a)\right] := \zeta.
\end{aligned}$$

Similar results can be derived for multi-step and iterative algorithms by defining $\hat{\pi}_0 = \hat{\pi}_\beta$*. With probability* $1-\delta$*,*

$$J(\hat{\pi}_T) - J(\hat{\pi}_\beta) = \sum_{t=1}^{T} J(\hat{\pi}_t) - J(\hat{\pi}_{t-1}) \geq \sum_{t=1}^{T} \zeta^{(t)},$$

where $\mathcal{D}(s,a)$ *denotes number of samples at* (s,a)*,* $C_{\gamma,\delta}$ *denotes the learning coefficient of SARSA and* $C_{\text{CFPI}}(s,a)$ *denotes the first-order approximation error from* (3.1)*. We defer detailed derivation and the expression of* $C_{\gamma,\delta}, \zeta^{(t)}$ *and* $C_{\text{CFPI}}(s,a)$ *in subsection 3.5.4. Note that when* $a = a_\beta$*,* $C_{\text{CFPI}}(s,a) = 0$*.*

By Theorem 3.2.2, $\hat{\pi}_1$ is a ζ-safe improved policy. The ζ safeness consists of two parts: C_{CFPI} is caused by the first-order approximation, and the $C_{\gamma,\delta}/\sqrt{\mathcal{D}(s,a)}$ term is incurred by the SARSA update. Similarly, $\hat{\pi}_T$ is a $\sum_{t=1}^{T}\zeta^{(t)}$-safe improved policy.

3.3 Related Work

Our methods belong and are motivated by the successful BCPO paradigm, which imposes constraints as in (2.2) to prevent from selecting OOD actions. Algorithms from this paradigm may apply different divergence functions, e.g., KL-divergence [11, 13], Rényi divergence [14, 15], MMD [9] or the MSE [10]. All these methods perform policy improvement via SGD. Instead, we perform CFPI by solving a linear approximation of

(2.2). Another line of research enforces the behavior constraint via parameterization. BCQ [4] learns a generative model as the behavior policy and a Q function to select the action from a set of perturbed behavior actions. [33] further show that the perturbation model can be discarded.

The design of our CFPI operators is inspired by the SOTA online RL algorithm OAC [51], which treats the single Gaussian evaluation policy as the baseline π_b and obtains an optimistic exploration policy by solving a similar optimization problem as (3.5). Since the underlying action distribution of an offline dataset often exhibits multimodality, we extend the result to accommodate a Gaussian Mixture π_b and overcome additional optimization difficulties by leveraging Lemma 3.2.1. Importantly, we successfully incorporate our CFPI operators into an iterative algorithm (Sec. 3.4.1). When updating the critics, we construct the TD target with actions chosen by the policy improved by our CFPI operators. This is in stark contrast to OAC, as OAC only employs the exploration policy to collect new transitions from the environment and does not use the actions generated by the exploration policy to construct the TD target for the critic training. These key differences highlight the novelty of our proposed method. We defer additional comparisons between our methods and OAC to subsection 3.3.1. In subsection 3.3.2, we further draw connections with prior works that leveraged the Taylor expansion to RL.

Recently, one-step [52, 12] algorithms have achieved great success. Instead of iteratively performing policy improvement and evaluation, these methods only learn a Q function via SARSA without bootstrapping from OOD action value. These methods further apply an policy improvement operator [11, 53] to extract a policy. We also instantiate a one-step algorithm with our CFPI operator and evaluate on standard benchmarks.

3.3.1 Detailed comparison with OAC

While Equation 3.4 yields the same action as the mean of the single Gaussian exploration policy in OAC's Proposition 1, it is crucial to note that Equation (6) only provides the closed-form solution for our $\mathcal{I}_{\text{SG}}$, which assumes a single Gaussian baseline policy π_b. The single Gaussian assumption of π_b often fails to capture the full complexity and multimodality of the underlying action distribution of offline datasets, which motivates the development of our $\mathcal{I}_{\text{MG}}$. In contrast, OAC exclusively addresses a single Gaussian baseline policy.

When generalizing π_b from a single Gaussian to a Gaussian Mixture, the optimization problem (3.3) transforms into the optimization problem (3.8), resulting in a non-convex constraint that breaks the tractability. Consequently, we must address the additional optimization challenges. Our approach involves utilizing inequalities in Lemma 3.2.1, ultimately leading to our CFPI operator $\mathcal{I}_{\text{MG}}$, which can handle both single-mode and multimodal π_b. Experiment results indicate that the one-step algorithm instantiated by our $\mathcal{I}_{\text{MG}}$ significantly outperforms the single Gaussian version, $\mathcal{I}_{\text{SG}}$, and other baselines, as shown in Table 3.4 and Figure 3.1.

Moreover, our proposed CFPI operator can be incorporated into an iterative algorithm (shown in Table 3.1) that solves the policy improvement step in each training iteration and updates the critics with the TD target constructed from the actions chosen by the policy improved through our $\mathcal{I}_{\text{MG}}$. In contrast, OAC only utilizes the exploration policy from their Proposition 1 to gather new transitions from the environment and does not use the actions generated by the exploration policy to construct the TD target for the critic training. This further highlights the novelty of our proposed method.

In summary, while there are certainly similarities between our $\mathcal{I}_{\text{SG}}$ and OAC's Proposition 1, we contend that our proposed CFPI operator $\mathcal{I}_{\text{MG}}$ a significant advancement,

particularly in its capability to accommodate both single-mode and multimodal behavior policies.

3.3.2 Connection with prior works that leveraged the Taylor expansion to RL

There has been a history of leveraging the Taylor expansion to construct efficient RL algorithms. [49] proposed the conservative policy iteration that optimizes a mixture of policies towards its policy objective's lower bound, which is constructed by performing first-order Taylor expansion on the mixture coefficient. Later, SOTA deep RL algorithms TRPO [54] and PPO [55] extend the results to work with trust region policy constraints and learn a stochastic policy parameterized by a neural network. More recently, [56] developed a second-order Taylor expansion approach under similar online RL settings.

At a high level, both our works and previous methods propose to create a surrogate of the original policy objective by leveraging the Taylor expansion approach. However, our motivation to use Taylor expansion is fundamentally different from the previous works [49, 54, 55, 56], which leverage the Taylor expansion to construct a lower bound of the policy objective so that optimizing towards the lower bound translates into guaranteed policy improvement. However, these methods do not result in a closed-form solution to the policy and still require iterative policy updates.

On the other hand, our method leverages the Taylor expansion to construct a linear approximation of the policy objective, enabling the derivation of a closed-form solution to the policy improvement step and thus avoiding performing policy improvement via SGD. We highlight that our closed-form policy update cannot be possible without directly optimizing the parameter of the policy distribution. In particular, the parameter should belong to the action space. **We note that this is a significant conceptual difference**

between our method and previous works.

Specifically, PDL [49] parameterizes the mixture coefficient of a mixture policy as θ. TRPO [54] and PPO [55] set θ as the parameter of a neural network that outputs the parameters of a Gaussian distribution. In contrast, our methods learn deterministic policy $\pi(s) = \text{Dirac}(\theta(s))$ and directly optimize the parameter $\theta(s)$. We aim to learn a greedy π by solving $\theta(s) = \arg\max_a Q(s, a)$. However, obtaining a greedy π in continuous control is problematic [57]. Given the requirement of limited distribution shift in the offline RL, we thus leverage the first-order Taylor expansion to relax the problem into a more tractable form

$$\theta(s) = \arg\max_a \bar{Q}(s, a; a_\beta), \text{ s.t. } -\log \pi_\beta(a|s) \leq \delta, \tag{3.14}$$

where $\bar{Q}$ is defined in Equation 3.1. By modeling π_β as a Single Gaussian or Gaussian Mixture, we further transform the problem into a QCLP and thus derive the closed-form solution.

Finally, we note that both the trust region methods TRPO and PPO and our methods constrain the divergence between the learned policy and behavior policy. However, the behavior policy always remains unchanged in our offline RL settings. As TRPO and PPO are designed for the online RL tasks, the updated policy will be used to collect new data and becomes the new behavior policy in future training iteration.

3.4 Experiments

Our experiments aim to demonstrate the effectiveness of our CFPI operators. Firstly, on the standard offline RL benchmark D4RL [45], we show that instantiating offline RL algorithms with our CFPI operators in both one-step and iterative manners outperforms

Table 3.1: Comparison between our one-step policy and SOTA methods on the Gym-MuJoCo domain of D4RL. Our method uses the same τ for all datasets except Hopper-M-E (detailed in subsection 3.10.1). We report the mean and standard deviation of our method's performance across 10 seeds. Each seed contains an individual training process and evaluates the policy for 100 episodes. We use Cheetah for HalfCheetah, M for Medium, E for Expert, and R for Replay. We bold the best results for each task.

Dataset	SG-BC	MG-BC	DT	TT	OnestepRL	TD3+BC	CQL	IQL	Our $\mathcal{I}_{\text{MG}}(\hat{\pi}_\beta, \hat{Q}_0)$
Cheetah-M-v2	40.6	40.6	42.6	46.9	**55.6**	48.3	44.0	47.4	52.1 ± 0.3
Hopper-M-v2	53.7	53.9	67.6	61.1	83.3	59.3	58.5	66.2	$\mathbf{86.8 \pm 4.0}$
Walker2d-M-v2	71.9	70.0	74.0	79.8	85.6	83.7	72.5	78.3	$\mathbf{88.3 \pm 1.6}$
Cheetah-M-R-v2	34.9	33.0	36.6	41.9	42.4	44.6	**45.5**	44.2	44.5 ± 0.4
Hopper-M-R-v2	12.4	21.2	82.7	91.5	71.0	60.9	**95.0**	94.7	93.6 ± 7.9
Walker2d-M-R-v2	22.9	22.8	66.6	**82.6**	71.6	81.8	77.2	73.8	78.2 ± 5.6
Cheetah-M-E-v2	46.6	51.7	86.8	95.0	93.5	90.7	91.6	86.7	$\mathbf{97.3 \pm 1.8}$
Hopper-M-E-v2	53.9	69.2	**107.6**	101.9	102.1	98.0	105.4	91.5	104.2 ± 5.1
Walker2d-M-E-v2	92.3	93.2	108.1	110.0	**110.9**	110.1	108.8	109.6	$\mathbf{111.9 \pm 0.3}$
Total	429.1	455.6	672.6	710.1	716.0	677.4	698.5	692.4	$\mathbf{757.0 \pm 27.0}$

SOTA methods (Sec. 3.4.1). Secondly, we show that our operators can improve a policy learned by other algorithms (Sec. 3.4.2). Ablation studies in Sec. 3.4.3 further show our superiority over the other policy improvement operators and demonstrate the benefit of modeling the behavior policy as a Gaussian Mixture.

3.4.1 Comparison with SOTA offline RL algorithms

We instantiate a one-step offline RL algorithm from Algorithm 1 with our policy improvement operator $\mathcal{I}_{\text{MG}}$. We learned a Gaussian Mixture baseline policy $\hat{\pi}_\beta$ via behavior cloning. We employed the IQN [58] architecture to model the Q value network for its better generalizability, as we need to estimate out-of-buffer $Q(s, a)$ during policy deployment. We trained the $\hat{Q}_0$ with SARSA algorithm [1, 59]. subsection 3.10.1 includes detailed training procedures of $\hat{\pi}_\beta$ and $\hat{Q}_0$ with full HP settings. We obtain our one-step policy as $\mathcal{I}_{\text{MG}}(\hat{\pi}_\beta, \hat{Q}_0; \tau)$.

We evaluate the effectiveness of our one-step algorithm on the D4RL benchmark

Table 3.2: Comparison between our Iterative $\mathcal{I}_{\text{MG}}$ and SOTA methods on the AntMaze domain. We report the mean and standard deviation across 5 seeds for our method with each seed evaluating for 100 episodes. The performance for all baselines is directly reported from the IQL paper. Our Iterative $\mathcal{I}_{\text{MG}}$ outperforms all baselines on 5 out of 6 tasks and obtains the best overall performance.

Dataset	BC	DT	Onestep RL	TD3+BC	CQL	IQL	Iterative $\mathcal{I}_{\text{MG}}$
antmaze-umaze-v0	54.6	59.2	64.3	78.6	74.0	87.5	$\mathbf{90.2 \pm 3.9}$
antmaze-umaze-diverse-v0	45.6	49.3	60.7	71.4	**84.0**	62.2	58.6 ± 15.2
antmaze-medium-play-v0	0.0	0.0	0.3	10.6	61.2	71.2	$\mathbf{75.2 \pm 6.9}$
antmaze-medium-diverse-v0	0.0	0.7	0.0	3.0	53.7	70.0	$\mathbf{72.2 \pm 7.3}$
antmaze-large-play-v0	0.0	0.0	0.0	0.2	15.8	39.6	$\mathbf{51.4 \pm 7.7}$
antmaze-large-diverse-v0	0.0	1.0	0.0	0.0	14.9	47.5	$\mathbf{52.4 \pm 10.9}$
Total	100.2	112.2	125.3	163.8	303.6	378.0	$\mathbf{400.0 \pm 52.0}$

focusing on the Gym-MuJoCo domain, which contains locomotion tasks with dense rewards. Table 3.1 compares our one-step algorithm with SOTA methods, including the other one-step actor-critic methods IQL [52], OneStepRL [12], BCPO method TD3+BC [10], conservative method CQL [37], and trajectory optimization methods DT [60], TT [61]. We also include the performance of two behavior policies SG-BC and MG-BC modeled with Single Gaussian and Gaussian Mixture, respectively. We directly report results for IQL, BCQ, TD3+BC, CQL, and DT from the IQL paper, and TT's result from its own paper. Note that OneStepRL instantiates three different algorithms. We only report its (Rev. KL Reg) result because this algorithm follows the BCPO paradigm and achieves the best overall performance. We highlight that OnesteRL reports the results by tuning the HP for each dataset.

Results in Table 3.1 demonstrate that our one-step algorithm outperforms the other algorithms by a significant margin without training a policy to maximize its Q-value through SGD. We note that we use the same τ for all datasets except Hopper-M-E. In Sec. 3.4.3, we will perform ablation studies and provide a fair comparison between our CFPI operators and the other policy improvement operators.

We further instantiate an iterative algorithm with $\mathcal{I}_{\text{MG}}$ and evaluate its effectiveness

on the challenging AntMaze domain of D4RL. The 6 tasks from AntMaze are more challenging due to their sparse-reward nature and lack of optimal trajectories in the static datasets. Table 3.2 compares our Iterative $\mathcal{I}_{\text{MG}}$ with SOTA algorithms on the AntMaze domain. Our method uses the same set of HP for all 6 tasks, outperforming all baselines on 5 out of 6 tasks and obtaining the best overall performance. subsection 3.7.1 presents additional details with training curves and pseudo-codes.

3.4.2 Improvement over a learned policy

Table 3.3: Our $\mathcal{I}_{\text{SG}}(\pi_{\text{IQL}}, Q_{\text{IQL}})$ improves the IQL policy π_{IQL} on AntMaze. We report the mean and standard deviation of 10 seeds. Each seed evaluates for 100 episodes.

Dataset	π_{IQL} (train)	π_{IQL} (1M)	$\mathcal{I}_{\text{SG}}(\pi_{\text{IQL}}, Q_{\text{IQL}})$
antmaze-u-v0	$\mathbf{87.4 \pm 3.2}$	83.6 ± 3.2	85.1 ± 5.3
antmaze-u-d-v0	$\mathbf{59.0 \pm 5.7}$	55.8 ± 7.9	55.0 ± 9.1
antmaze-m-p-v0	71.1 ± 5.43	64.2 ± 13.2	$\mathbf{75.5 \pm 6.1}$
antmaze-m-d-v0	70.0 ± 6.16	66.8 ± 9.4	$\mathbf{79.9 \pm 3.8}$
antmaze-l-p-v0	34.4 ± 6.04	35.6 ± 7.0	$\mathbf{37.7 \pm 7.7}$
antmaze-l-d-v0	39.8 ± 9.09	38.8 ± 7.1	$\mathbf{40.1 \pm 5.6}$
Total	361.7 ± 35.6	344.7 ± 47.8	$\mathbf{373.3 \pm 37.5}$

In this section, we show that our CFPI operator $\mathcal{I}_{\text{SG}}$ can further improve the performance of a Single Gaussian policy π_{IQL} learned by IQL [52] on the AntMaze domain. We first obtain the IQL policy π_{IQL} and Q_{IQL} by training for 1M gradient steps using the PyTorch Implementation from RLkit [62]. We emphasize that we follow the authors' exact training and evaluation protocols and include all training curves in subsection 3.11.6. Interestingly, even though the average of the evaluation results during training matches the results reported in the IQL paper, Table 3.3 shows that the evaluation of the final 1M-step policy π_{IQL} does not match the reported performance on all 6 tasks. This demonstrates how drastically performance can fluctuate across just dozens of epochs, echoing the unstable performance of offline-trained policies highlighted by [10]. Thanks to the

Table 3.4: Ablation studies of our Method on the Gym-MuJoCo domain. Again we report the mean and standard deviation of 10 seeds, and each seed evaluates for 100 episodes. Our $\mathcal{I}_{\text{MG}}$ outperforms baselines by a significant margin. At the same time, the SGD-based method Rev. KL Reg exhibits substantial performance variations, demonstrating the importance of a stable policy improvement operator.

Dataset	SG-EBCQ	MG-EBCQ	SG-Rev. KL Reg	MG-Rev. KL Reg	$\mathcal{I}_{\text{SG}}$	$\mathcal{I}_{\text{MG}}$
Cheetah-M-v2	$\mathbf{53.3 \pm 0.2}$	51.5 ± 0.2	47.1 ± 0.2	47.0 ± 0.2	51.1 ± 0.1	52.1 ± 0.3
Hopper-M-v2	$\mathbf{86.8 \pm 5.2}$	82.5 ± 1.9	70.3 ± 7.0	76.3 ± 6.9	75.6 ± 3.7	$\mathbf{86.8 \pm 4.0}$
Walker2d-M-v2	85.2 ± 5.1	85.2 ± 2.1	82.4 ± 1.0	82.8 ± 1.8	88.1 ± 1.1	$\mathbf{88.3 \pm 1.6}$
Cheetah-M-R-v2	43.5 ± 0.6	43.0 ± 0.3	44.3 ± 0.4	44.4 ± 0.5	42.8 ± 0.4	$\mathbf{44.5 \pm 0.4}$
Hopper-M-R-v2	88.5 ± 12.2	83.6 ± 10.3	$\mathbf{99.7 \pm 1.0}$	99.4 ± 2.1	87.7 ± 8.7	93.6 ± 7.9
Walker2d-M-R-v2	75.4 ± 4.6	73.1 ± 5.2	63.6 ± 28.5	69.7 ± 30.9	71.3 ± 4.4	$\mathbf{78.2 \pm 5.6}$
Cheetah-M-E-v2	81.8 ± 5.4	84.5 ± 4.6	78.9 ± 9.8	65.0 ± 10.1	91.1 ± 3.1	$\mathbf{97.3 \pm 1.8}$
Hopper-M-E-v2	40.0 ± 5.8	56.1 ± 6.2	76.6 ± 18.3	$\mathbf{79.4 \pm 32.6}$	70.3 ± 8.9	73.0 ± 10.5
Walker2d-M-E-v2	111.1 ± 1.8	111.1 ± 1.0	106.7 ± 4.1	107.1 ± 4.0	111.1 ± 1.1	$\mathbf{111.9 \pm 0.3}$
Total	665.5 ± 41.0	670.6 ± 31.9	669.7 ± 70.3	671.2 ± 89.1	688.9 ± 31.6	$\mathbf{725.8 \pm 32.4}$

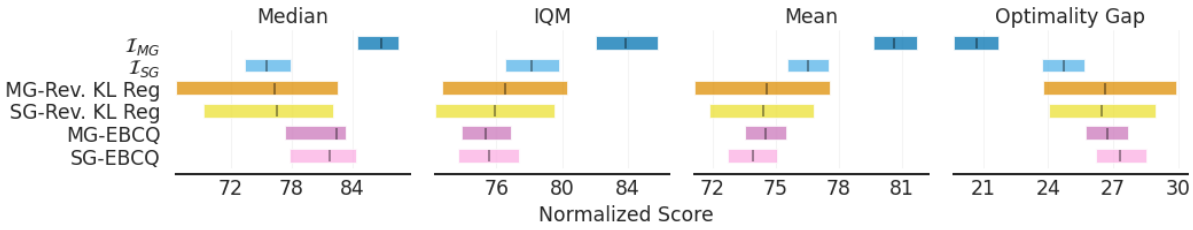


Figure 3.1: Aggregate metrics [63] with 95% CIs based on results reported in Table 3.4. The CIs are estimated using the percentile bootstrap with stratified sampling. Higher median, IQM, and mean scores, and lower Optimality Gap correspond to better performance. Our $\mathcal{I}_{\text{MG}}$ outperforms baselines by a significant margin based on all four metrics. section 3.9 includes additional details.

tractability of $\mathcal{I}_{\text{SG}}$, we directly obtain an improved policy $\mathcal{I}_{\text{SG}}(\pi_{\text{IQL}}, Q_{\text{IQL}}; \tau)$ that achieves better overall performance than both π_{IQL} (train) and (1M), as shown in Table 3.3. We tune the HP τ using a small set of seeds for each task following the practice of [12, 45] and include more details in subsection 3.10.2 and subsection 3.11.6.

3.4.3 Ablation studies

We first provide a fair comparison with the other policy improvement operators, demonstrating the effectiveness of solving the approximated BCPO (3.2) and modeling

the behavior policy as a Gaussian Mixture. Additionally, we examine the sensitivity on τ, ablate the number of Gaussian components, and discuss the limitation by ablating the Q network in subsection 3.11.2, subsection 3.11.3, and subsection 3.11.4, respectively.

Effectiveness of our CFPI operators. In Table 3.4, we compare our CFPI operators with two policy improvement operators, namely, Easy BCQ (EBCQ) and Rev. KL Reg from OneStepRL [12]. EBCQ doe not require training either, returning a policy by selecting an action that maximizes a learned $\hat{Q}$ from N_{bcq} actions randomly sampled from the behavior policy $\hat{\pi}_\beta$. Rev. KL Reg sets $D(\cdot, \cdot)$ in (2.2) as the reverse KL divergence and solves the problem via SGD, with α controlling the regularization strength. We omit the comparison with the other learning-based operator Exp. Weight, as Rev. KL Reg achieves the best overall performance in OneStepRL.

For all methods, we present results with $\hat{\pi}_\beta$ modeled by Single Gaussian (SG-) and Gaussian Mixture (MG-). To ensure a fair comparison, we employ the same $\hat{Q}_0$ and $\hat{\pi}_\beta$ modeled and learned in the same way as in Sec. 3.4.1 for all methods. Moreover, we tune N_{bcq} for EBCQ, α for Rev. KL Reg, and τ for our methods. Each method uses the same set of HP for all datasets. As a result, the Hopper-M-E performance of $\mathcal{I}_{\text{MG}}$ reported in Table 3.4 is different from Table 3.1. subsection 3.10.1 includes details on the HP tuning and corresponding experiment results in Table 3.9, 3.10, 3.11 and 3.12.

As is shown in Table 3.4 and Fig. 3.1, our $\mathcal{I}_{\text{MG}}$ clearly outperforms all baselines by a significant margin. The SGD-based method Rev. KL Reg exhibits substantial performance variations, highlighting the need for designing stable policy improvement operators in offline RL. Moreover, our CFPI operators outperform their EBCQ counterparts, demonstrating the effectiveness of solving the approximated BCPO.

Effectiveness of Gaussian Mixture. As the three M-E datasets are collected by an expert and medium policy, we should recover the expert performance if we can 1) capture the two modes of the action distribution 2) and always select action from the expert

mode. In other words, we can leverage the $\hat{Q}_0$ learned by SARSA to select actions from the mean of each Gaussian component, resulting in a mode selection algorithm (MG-MS) that selects its action by

$$\begin{aligned} \mu_{\text{mode}} = \arg\max_{\hat{\mu}_i} \quad & \hat{Q}_0(s, \hat{\mu}_i), \\ \text{s.t.} \quad & \{\hat{\mu}_i | \hat{\lambda}_i > \xi\}, \ \sum\nolimits_{i=1:N} \hat{\lambda}_i \mathcal{N}(\hat{\mu}_i, \hat{\Sigma}_i) = \hat{\pi}_\beta, \end{aligned} \tag{3.15}$$

ξ is set to filter out trivial components. Our MG-MS achieves an expert performance on Hopper-M-E (104.2 ± 5.1) and Walker2d-M-E (104.1 ± 6.7) and matches SOTA algorithms in Cheetah-M-E (91.3 ± 2.1). subsection 3.11.1 includes full results of MG-MS on the Gym MuJoCo domain.

Table 3.4 show that with the same $\hat{Q}_0$, $\mathcal{I}_{\text{MG}}$ with Gaussian Mixture $\hat{\pi}_\beta$ outperforms $\mathcal{I}_{\text{SG}}$ with Single Gaussian $\hat{\pi}_\beta$ on all M-R and M-E datasets. The $\hat{Q}_0$ is modeled and learned in the same way as in Sec. 3.4.1. Here, we slightly abuse the notation and learn both $\hat{\pi}_\beta$ via behavior cloning. We highlight that we tune τ separately for $\mathcal{I}_{\text{MG}}$ and $\mathcal{I}_{\text{SG}}$ and each method use the same set of HP for all 9 datasets. As a result, the Hopper-M-E performance of $\mathcal{I}_{\text{MG}}$ reported in Table 3.4 is different from Table 3.1.

Our experiments are conducted on various types of 8GPUs machines. Different machines may have different GPU types, such as NVIDIA GA100 and TU102. Training a behavior policy for 500K gradient steps takes around 40 minutes, while training a Q network for 500K gradient steps takes around 50 minutes.

3.5 Proofs and Theoretical Results

3.5.1 Proof of Proposition 3.2.1

Proposition 3.5.1. The optimization problem (3.3) has a closed-form solution that is given by

$$\mu_{\text{sg}}(\tau) = \mu_\beta + \frac{\sqrt{2\log\tau}\,\Sigma_\beta\left[\nabla_a Q(s,a)\right]_{a=\mu_\beta}}{\left\|\left[\nabla_a Q(s,a)\right]_{a=\mu_\beta}\right\|_{\Sigma_\beta}}, \text{ where } \delta = \frac{1}{2}\log\det(2\pi\Sigma_\beta) + \log\tau \quad (3.16)$$

Proof. The optimization problem (3.3) can be converted into the QCLP

$$\max_{\mu} \quad \mu^T\left[\nabla_a Q(s,a)\right]_{a=\mu_\beta}, \quad \text{s.t.} \quad \frac{1}{2}(\mu-\mu_\beta)^T\Sigma_\beta^{-1}(\mu-\mu_\beta) \le \delta - \frac{1}{2}\log\det(2\pi\Sigma_\beta) \quad (3.17)$$

Following a similar procedure as is in OAC [51], we first derive the Lagrangian below:

$$L = \mu^T\left[\nabla_a Q(s,a)\right]_{a=\mu_\beta} - \eta\left(\frac{1}{2}(\mu-\mu_\beta)^T\Sigma_\beta^{-1}(\mu-\mu_\beta) - \delta + \frac{1}{2}\log\det(2\pi\Sigma_\beta)\right) \quad (3.18)$$

Taking the derivatives w.r.t μ, we get

$$\nabla_\mu L = \left[\nabla_a Q(s,a)\right]_{a=\mu_\beta} - \eta\Sigma_\beta^{-1}(\mu-\mu_\beta) \quad (3.19)$$

By setting $\nabla_\mu L = 0$, we get

$$\mu = \mu_\beta + \frac{1}{\eta}\Sigma_\beta\left[\nabla_a Q(s,a)\right]_{a=\mu_\beta} \quad (3.20)$$

To satisfy the the KKT conditions [46], we have $\eta > 0$ and

$$(\mu - \mu_\beta)^T \Sigma_\beta^{-1} (\mu - \mu_\beta) = 2\delta - \log\det(2\pi\Sigma_\beta) \tag{3.21}$$

Finally with (3.20) and (3.21), we get

$$\eta = \sqrt{\frac{\left[\nabla_a Q(s,a)\right]_{a=\mu_\beta}^T \Sigma_\beta \left[\nabla_a Q(s,a)\right]_{a=\mu_\beta}}{2\delta - \log\det(2\pi\Sigma_\beta)}} \tag{3.22}$$

By setting $\delta = \frac{1}{2}\log\det(2\pi\Sigma_\beta) + \log\tau$ and plugging (3.22) to (3.20), we obtain the final solution as

$$\begin{aligned} \mu_{\text{sg}}(\tau) &= \mu_\beta + \frac{\sqrt{2\log\tau}\,\Sigma_\beta \left[\nabla_a Q(s,a)\right]_{a=\mu_\beta}}{\sqrt{\left[\nabla_a Q(s,a)\right]_{a=\mu_\beta}^T \Sigma_\beta \left[\nabla_a Q(s,a)\right]_{a=\mu_\beta}}}, \\ &= \mu_\beta + \frac{\sqrt{2\log\tau}\,\Sigma_\beta \left[\nabla_a Q(s,a)\right]_{a=\mu_\beta}}{\left\| \left[\nabla_a Q(s,a)\right]_{a=\mu_\beta} \right\|_{\Sigma_\beta}} \end{aligned} \tag{3.23}$$

which completes the proof. □

3.5.2 Proof of Proposition 3.2.2

Proposition 3.2. By applying the first inequality of Lemma 3.2.1 to the constraint of (3.8), we can derive an optimization problem that lower bounds (3.8)

$$\begin{aligned} \max_{\mu} \quad & \mu^T \left[\nabla_a Q(s,a)\right]_{a=a_\beta} \\ \text{s.t.} \quad & \max_i \left\{ -\frac{1}{2}(\mu-\mu_i)^T \Sigma_i^{-1}(\mu-\mu_i) - \frac{1}{2}\log\det(2\pi\Sigma_i) + \log\lambda_i \right\} \geq -\delta, \end{aligned} \tag{3.24}$$

and the closed-form solution to (3.9) is given by

$$\begin{aligned} \mu_{\text{lse}}(\tau) = \arg\max_{\overline{\mu}_i(\delta)} \quad & \overline{\mu}_i{}^T \left[\nabla_a Q(s,a)\right]_{a=\mu_i}, \quad \text{s.t.} \quad \delta = \frac{1}{2}\min_i \left\{\log\lambda_i \det(2\pi\Sigma_i)\right\} + \log\tau \\ \text{where} \quad \overline{\mu}_i(\delta) = \mu_i + & \frac{\kappa_i \Sigma_i \left[\nabla_a Q(s,a)\right]_{a=\mu_i}}{\left\| \left[\nabla_a Q(s,a)\right]_{a=\mu_i} \right\|_{\Sigma_i}}, \quad \text{and} \quad \kappa_i = \sqrt{2(\delta+\log\lambda_i) - \log\det(2\pi\Sigma_i)}. \end{aligned} \tag{3.25}$$

Proof. Recall that the Gaussian Mixture behavior policy is constructed by

$$\pi_\beta = \sum_{i=1}^{N} \lambda_i \mathcal{N}(\mu_i, \Sigma_i), \tag{3.26}$$

We first divide the optimization problem (3.24) into N sub-problems, with each sub-problem i given by

$$\begin{aligned} \max_{\mu} \quad & \mu^T \left[\nabla_a Q(s,a)\right]_{a=a_\beta} \\ \text{s.t.} \quad & -\frac{1}{2}(\mu-\mu_i)^T \Sigma_i^{-1}(\mu-\mu_i) - \frac{1}{2}\log\det(2\pi\Sigma_i) + \log\lambda_i \geq -\delta, \end{aligned} \tag{3.27}$$

which is equivalent to solving problem (3.3) for each Gaussian component with an additional constant term $\log\lambda_i$, and thus has a *unique* closed-form solution.

Define the maximizer for each sub-problem i as $\overline{\mu}_i(\delta)$, though $\overline{\mu}_i(\delta)$ does not always exist. Whenever $-\frac{1}{2}\log\det(2\pi\Sigma_i)+\log\lambda_i < -\delta$, there will be no μ satisfying the constraint

as $\frac{1}{2}(\mu-\mu_i)^T\Sigma_i^{-1}(\mu-\mu_i)$ is always greater than 0. We thus set $\overline{\mu}_i(\delta)$ to be *None* in this case. Next, we will show that there does not exist any $\breve{\mu} \notin \{\overline{\mu}_i(\delta)|i=1\dots N\}$, s.t., $\breve{\mu}$ is the maximizer of (3.24). We can show this by contradiction. Suppose there exists a $\breve{\mu} \notin \{\overline{\mu}_i(\delta)|i=1\dots N\}$ maximizing (3.24), there exists at least one $j \in \{1,\dots,N\}$ s.t.

$$-\frac{1}{2}(\breve{\mu}-\mu_j)^T\Sigma_j^{-1}(\breve{\mu}-\mu_j) - \frac{1}{2}\log\det(2\pi\Sigma_j) + \log\lambda_j \geq -\delta. \tag{3.28}$$

Since $\breve{\mu}$ is the maximizer of (3.24), it should also be maximizer of the sub-problem j. However, the maximizer for sub-problem j is given by $\overline{\mu}_j(\delta) \neq \breve{\mu}$, contradicting with the fact that $\breve{\mu}$ is the maximizer of the sub-problem j. Therefore, the optimal solution to (3.24) has to be given by

$$\arg\max_{\overline{\mu}_i} \quad \overline{\mu}_i{}^T\left[\nabla_a Q(s,a)\right]_{a=a_\beta} \quad \text{where} \quad \overline{\mu}_i \in \{\overline{\mu}_i(\delta)|i=1\dots N\} \tag{3.29}$$

To solve each sub-problem i, it is natural to set $a_\beta = \mu_i$, which reformulate the sub-problem i as below

$$\begin{aligned} &\max_{\mu} \quad \mu^T\left[\nabla_a Q(s,a)\right]_{a=\mu_i} \\ &\text{s.t.} \quad \frac{1}{2}(\mu-\mu_i)^T\Sigma_i^{-1}(\mu-\mu_i) \leq \delta - \frac{1}{2}\log\det(2\pi\Sigma_i) + \log\lambda_i, \end{aligned} \tag{3.30}$$

Note that problem (3.30) is also a QCLP similar to the problem (3.3). Therefore, we can derive its solution by following similar procedures as in subsection 3.5.1, resulting in

$$\overline{\mu}_i(\delta) = \mu_i + \frac{\kappa_i\Sigma_i\left[\nabla_a Q(s,a)\right]_{a=\mu_i}}{\left\|\left[\nabla_a Q(s,a)\right]_{a=\mu_i}\right\|_{\Sigma_i}}, \quad \text{where} \quad \kappa_i = \sqrt{2(\delta+\log\lambda_i) - \log\det(2\pi\Sigma_i)}. \tag{3.31}$$

We complete the proof by further setting $\delta = \frac{1}{2}\min_i\{\log\lambda_i\det(2\pi\Sigma_i)\} + \log\tau$. □

3.5.3 Proof of Proposition 3.2.3

Proposition 3.3. By applying the second inequality of Lemma 3.2.1 to the constraint of (3.8), we can derive an optimization problem that lower bounds (3.8)

$$\begin{aligned} \max_{\mu} \quad & \mu^T \left[\nabla_a Q(s,a)\right]_{a=a_\beta} \\ \text{s.t.} \quad & \sum_{i=1}^{N} \lambda_i \left(-\frac{1}{2}\log\det(2\pi\Sigma_i) - \frac{1}{2}(\mu-\mu_i)^T\Sigma_i^{-1}(\mu-\mu_i) \right) \geq -\delta \end{aligned} \tag{3.32}$$

and the closed-form solution to (3.11) is given by

$$\begin{aligned} \mu_{\text{jensen}}(\tau) = \overline{\mu} + \frac{\kappa_i \overline{\Sigma}[\nabla_a Q(s,a)]_{a=\overline{\mu}}}{\left\| [\nabla_a Q(s,a)]_{a=\overline{\mu}} \right\|_{\overline{\Sigma}}}, \quad \text{where} \quad \kappa_i = \sqrt{2\log\tau - \sum_{i=1}^{N} \lambda_i \mu_i^T \Sigma_i^{-1} \mu_i + \overline{\mu}^T \overline{\Sigma}^{-1} \overline{\mu}}\ , \\ \overline{\Sigma} = \left(\sum_{i=1}^{N} \lambda_i \Sigma_i^{-1} \right)^{-1}, \quad \overline{\mu} = \overline{\Sigma} \left(\sum_{i=1}^{N} \lambda_i \Sigma_i^{-1} \mu_i \right), \quad \delta = \log\tau + \frac{1}{2}\sum_{i=1}^{N} \lambda_i \log\det(2\pi\Sigma_i) \end{aligned} \tag{3.33}$$

Proof. Note that problem (3.32) is also a QCLP. Before deciding the value of a_β, we first derive its Lagrangian with a general a_β below

$$L = \mu^T \left[\nabla_a Q(s,a)\right]_{a=a_\beta} - \eta \left(\sum_{i=1}^{N} \lambda_i \left(\frac{1}{2}\log\det(2\pi\Sigma_i) + \frac{1}{2}(\mu-\mu_i)^T\Sigma_i^{-1}(\mu-\mu_i) \right) - \delta \right) \tag{3.34}$$

Taking the derivatives w.r.t μ, we get

$$\nabla_\mu L = \left[\nabla_a Q(s,a)\right]_{a=a_\beta} - \eta \left(\sum_{i=1}^{N} \lambda_i \left(\Sigma_i^{-1}(\mu-\mu_i) \right) \right) \tag{3.35}$$

By setting $\nabla_\mu L = 0$, we get

$$\begin{aligned}\mu &= \left(\sum_{i=1}^{N} \lambda_i \Sigma_i^{-1}\right)^{-1} \left(\sum_{i=1}^{N} \lambda_i \Sigma_i^{-1} \mu_i\right) + \frac{1}{\eta}\left(\sum_{i=1}^{N} \lambda_i \Sigma_i^{-1}\right) \left[\nabla_a Q(s,a)\right]_{a=a_\beta} \\ &= \overline{\mu} + \frac{1}{\eta}\overline{\Sigma}\left[\nabla_a Q(s,a)\right]_{a=a_\beta}, \qquad (3.36) \\ \text{where} \quad \overline{\Sigma} &= \left(\sum_{i=1}^{N} \lambda_i \Sigma_i^{-1}\right)^{-1}, \quad \overline{\mu} = \overline{\Sigma}\left(\sum_{i=1}^{N} \lambda_i \Sigma_i^{-1} \mu_i\right),\end{aligned}$$

Equation 3.36 shows that the final solution to the problem (3.32) will be a shift from the pseudo-mean $\overline{\mu}$. Therefore, setting $a_\beta = \overline{\mu}$ becomes a natural choice.

Furthermore, by satisfying the KKT conditions, we have $\eta > 0$ and

$$\sum_{i=1}^{N} \lambda_i \left(\mu - \mu_i\right)^T \Sigma_i^{-1} \left(\mu - \mu_i\right) = 2\delta - \sum_{i=1}^{N} \lambda_i \log\det(2\pi\Sigma_i) \qquad (3.37)$$

Plugging (3.32) into (3.37) gives the equation below

$$\begin{aligned}&\sum_{i=1}^{N} \lambda_i \left(\overline{\mu} + \frac{1}{\eta}\overline{\Sigma}\left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}} - \mu_i\right)^T \Sigma_i^{-1} \left(\overline{\mu} + \frac{1}{\eta}\overline{\Sigma}\left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}} - \mu_i\right) \\ &\quad = 2\delta - \sum_{i=1}^{N} \lambda_i \log\det(2\pi\Sigma_i).\end{aligned} \qquad (3.38)$$

The LHS of (3.38) can be reformulated as

$$\begin{aligned}
&\sum_{i=1}^{N} \lambda_i \left(\overline{\mu} + \frac{1}{\eta} \overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} - \mu_i \right)^T \Sigma_i^{-1} \left(\overline{\mu} + \frac{1}{\eta} \overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} - \mu_i \right) \\
= \quad & \frac{1}{\eta^2} \sum_{i=1}^{N} \lambda_i \left(\overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} \right)^T \Sigma_i^{-1} \left(\overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} \right) \\
& + \frac{2}{\eta} \sum_{i=1}^{N} \lambda_i \left(\overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} \right)^T \Sigma_i^{-1} \left(\overline{\mu} - \mu_i \right) \\
& + \sum_{i=1}^{N} \lambda_i \left(\overline{\mu} - \mu_i \right)^T \Sigma_i^{-1} \left(\overline{\mu} - \mu_i \right)
\end{aligned} \quad . \tag{3.39}$$

We note that the second line of (3.39)'s RHS can be reduced to

$$\begin{aligned}
& \frac{2}{\eta} \sum_{i=1}^{N} \lambda_i \left(\overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} \right)^T \Sigma_i^{-1} \left(\overline{\mu} - \mu_i \right) \\
= \quad & \frac{2}{\eta} \left(\overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} \right)^T \left(\left(\sum_{i=1}^{N} \lambda_i \Sigma_i^{-1} \right) \overline{\mu} - \sum_{i=1}^{N} \lambda_i \Sigma_i^{-1} \mu_i \right) \\
= \quad & \frac{2}{\eta} \left(\overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} \right)^T \left(\overline{\Sigma}^{-1} \overline{\mu} - \overline{\Sigma}^{-1} \left(\overline{\Sigma} \sum_{i=1}^{N} \lambda_i \Sigma_i^{-1} \mu_i \right) \right) . \\
= \quad & \frac{2}{\eta} \left(\overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} \right)^T \left(\overline{\Sigma}^{-1} \overline{\mu} - \overline{\Sigma}^{-1} \overline{\mu} \right) \\
= \quad & 0
\end{aligned} \tag{3.40}$$

Therefore, (3.39) can be further reformulated as

$$\begin{aligned}
& \sum_{i=1}^{N} \lambda_i \left(\overline{\mu} + \frac{1}{\eta} \overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} - \mu_i \right)^T \Sigma_i^{-1} \left(\overline{\mu} + \frac{1}{\eta} \overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} - \mu_i \right) \\
= & \frac{1}{\eta^2} \sum_{i=1}^{N} \lambda_i \left(\overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} \right)^T \Sigma_i^{-1} \left(\overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} \right) \\
& + \sum_{i=1}^{N} \lambda_i \left(\overline{\mu} - \mu_i \right)^T \Sigma_i^{-1} \left(\overline{\mu} - \mu_i \right) \\
= & \frac{1}{\eta^2} \left(\overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} \right)^T \left(\sum_{i=1}^{N} \lambda_i \Sigma_i^{-1} \right) \left(\overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} \right) \\
& + \sum_{i=1}^{N} \lambda_i \left(\overline{\mu} - \mu_i \right)^T \Sigma_i^{-1} \left(\overline{\mu} - \mu_i \right) \\
= & \frac{1}{\eta^2} \left(\overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} \right)^T \overline{\Sigma}^{-1} \left(\overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} \right) \\
& + \sum_{i=1}^{N} \lambda_i \left(\overline{\mu} - \mu_i \right)^T \Sigma_i^{-1} \left(\overline{\mu} - \mu_i \right) \\
= & \frac{1}{\eta^2} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}}^T \overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} + \sum_{i=1}^{N} \lambda_i \left(\overline{\mu} - \mu_i \right)^T \Sigma_i^{-1} \left(\overline{\mu} - \mu_i \right)
\end{aligned} \quad . \tag{3.41}$$

To this point, (3.38) can be reformulated as

$$\begin{aligned}
& \frac{1}{\eta^2} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}}^T \overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}} + \sum_{i=1}^{N} \lambda_i \left(\overline{\mu} - \mu_i \right)^T \Sigma_i^{-1} \left(\overline{\mu} - \mu_i \right) \\
= & \; 2\delta - \sum_{i=1}^{N} \lambda_i \log \det(2\pi\Sigma_i)
\end{aligned} \tag{3.42}$$

We can thus express η as below

$$\eta = \sqrt{\frac{\left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}}^T \overline{\Sigma} \left[\nabla_a Q(s,a) \right]_{a=\overline{\mu}}}{2\delta - \sum_{i=1}^{N} \lambda_i \log \det(2\pi\Sigma_i) - \sum_{i=1}^{N} \lambda_i \left(\overline{\mu} - \mu_i \right)^T \Sigma_i^{-1} \left(\overline{\mu} - \mu_i \right)}} \tag{3.43}$$

By setting $\delta = \frac{1}{2}\sum_{i=1}^{N} \lambda_i \log\det(2\pi\Sigma_i) + \log\tau$, we have

$$\begin{aligned}
\eta &= \sqrt{\frac{\left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}}^T \overline{\Sigma} \left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}}}{2\log\tau - \sum_{i=1}^{N} \lambda_i \left(\overline{\mu} - \mu_i\right)^T \Sigma_i^{-1} \left(\overline{\mu} - \mu_i\right)}} \\
&= \sqrt{\frac{\left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}}^T \overline{\Sigma} \left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}}}{2\log\tau - \sum_{i=1}^{N} \lambda_i \overline{\mu}^T \Sigma_i^{-1} \overline{\mu} + 2\overline{\mu}^T \sum_{i=1}^{N} \lambda_i \Sigma_i^{-1} \mu_i - \sum_{i=1}^{N} \lambda_i \mu_i^T \Sigma_i^{-1} \mu_i}} \\
&= \sqrt{\frac{\left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}}^T \overline{\Sigma} \left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}}}{2\log\tau - \sum_{i=1}^{N} \overline{\mu}^T \overline{\Sigma}^{-1} \overline{\mu} + 2\overline{\mu}^T \overline{\Sigma}^{-1} \overline{\mu} - \sum_{i=1}^{N} \lambda_i \mu_i^T \Sigma_i^{-1} \mu_i}} \\
&= \sqrt{\frac{\left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}}^T \overline{\Sigma} \left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}}}{2\log\tau + \overline{\mu}^T \overline{\Sigma}^{-1} \overline{\mu} - \sum_{i=1}^{N} \lambda_i \mu_i^T \Sigma_i^{-1} \mu_i}}
\end{aligned} \tag{3.44}$$

Finally, plugging (3.44) into (3.36), with $a_\beta = \overline{\mu}$, we have

$$\begin{aligned}
\mu_{\text{jensen}}(\tau) &= \overline{\mu} + \sqrt{\frac{2\log\tau - \sum_{i=1}^{N} \lambda_i \mu_i^T \Sigma_i^{-1} \mu_i + \overline{\mu}^T \overline{\Sigma}^{-1} \overline{\mu}}{\left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}}^T \overline{\Sigma} \left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}}}}\ \overline{\Sigma} \left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}} \\
&= \overline{\mu} + \frac{\kappa_i \overline{\Sigma} \left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}}}{\left\| \left[\nabla_a Q(s,a)\right]_{a=\overline{\mu}} \right\|_{\overline{\Sigma}}}, \\
&\text{where} \quad \kappa_i = \sqrt{2\log\tau - \sum_{i=1}^{N} \lambda_i \mu_i^T \Sigma_i^{-1} \mu_i + \overline{\mu}^T \overline{\Sigma}^{-1} \overline{\mu}}
\end{aligned} \tag{3.45}$$

which completes the proof. □

3.5.4 Proof of Theorem 3.2.2

In this section, we prove the *safe policy improvement* presented in Section 3.2.3. Algorithm 1 follows the *approximate policy iteration* (API) [47] by iterating over the policy evaluation ($\mathcal{E}$ step, Line 4) and policy improvement ($\mathcal{I}$ step, Line 5). Therefore, to verify $\mathcal{E}$ provides the improvement, we need to first show policy evaluation $\hat{Q}_t$ is accurate. In particular, we focus on the SARSA updates (Line 2), which is a form of on-policy Fitted Q-Iteration [1]. Fortunately, it is known that FQI is statistically efficient (*e.g.* [48]) under the mild condition for the function approximation class. Its linear counterpart, least-square value iteration, is also shown to be efficient for offline reinforcement learning [64, 65]. Recently, [66] shows the finite sample convergence guarantee for SARSA under the standard the mean square error loss.

Next, to show the performance improvement, we leverage the performance difference lemma to show our algorithm achieves the desired goal.

Lemma 3.5.1 (Performance Difference Lemma)**.** *For any policy π, π', it holds that*

$$J(\pi) - J(\pi') = \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\pi}}\left[\mathbb{E}_{a\sim\pi(\cdot|s)}A^{\pi'}(s,a)\right],$$

where $A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s)$ is the advantage function.

Similar to [37], we focus on the discrete case where the number of states $|\mathcal{S}|$ and actions $|\mathcal{A}|$ are finite (note in the continuous case, the $\mathcal{D}(s,a)$ would be 0 for most locations, and thus the bound becomes less interesting). The adaptation to the continuous space can leverage standard techniques like *state abstraction* [67] and covering arguments.

Next, we define the learning coefficient $C_{\gamma,\delta}$ of SARSA as

$$|\hat{Q}^{\hat{\pi}_\beta}(s,a) - Q^{\hat{\pi}_\beta}(s,a)| \leq \frac{C_{\gamma,\delta}}{\sqrt{\mathcal{D}(s,a)}}, \quad \forall s,a \in \mathcal{S}\times\mathcal{A}.$$

Define the first-order approximation error as

$$\bar{Q}^{\hat{\pi}_\beta}(s,a) := (a - a_\beta)^T \left[\nabla_a \hat{Q}^{\hat{\pi}_\beta}(s,a)\right]_{a=a_\beta} + \hat{Q}^{\hat{\pi}_\beta}(s,a_\beta),$$

then the approximation error is defined as:

$$C_{\text{CFPI}}(s,a) := |\bar{Q}^{\hat{\pi}_\beta}(s,a) - \hat{Q}^{\hat{\pi}_\beta}(s,a)|.$$

Under the constraint $D\left(\pi(\cdot \mid s), \hat{\pi}_\beta(\cdot \mid s)\right) \leq \delta$ (3.2) (or equivalently action a is close to a_β), the first-order approximation provides a good estimation for the $\hat{Q}^{\pi_\beta}$.

Theorem 3.5.2 (Restatement of Theorem 3.2.2)**.** *Assume the state and action spaces are discrete. Let $\hat{\pi}_1$ be the policy obtained after the CFPI update (Line 2 of Algorithm 1). Then with probability $1-\delta$,*

$$\begin{aligned} J(\hat{\pi}_1) - J(\hat{\pi}_\beta) \geq & \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\hat{\pi}_1}}\left[\bar{Q}^{\hat{\pi}_\beta}(s,\hat{\pi}_1(s)) - \bar{Q}^{\hat{\pi}_\beta}(s,\hat{\pi}_\beta(s))\right] \\ & - \frac{2}{1-\gamma}\mathbb{E}_{s\sim d^{\hat{\pi}_1}}\mathbb{E}_{a\sim\hat{\pi}_1(\cdot|s)}\left[\frac{C_{\gamma,\delta}}{\sqrt{\mathcal{D}(s,a)}} + C_{\text{CFPI}}(s,a)\right] := \zeta. \end{aligned}$$

Similar results can be derived for multi-step and iterative algorithms by defining $\hat{\pi}_0 = \hat{\pi}_\beta$. With probability $1-\delta$,

$$J(\hat{\pi}_T) - J(\hat{\pi}_\beta) = \sum_{t=1}^{T} J(\hat{\pi}_t) - J(\hat{\pi}_{t-1}) \geq \sum_{t=1}^{T} \zeta^{(t)},$$

where $\mathcal{D}(s,a)$ denotes number of samples at s,a, the learning coefficient of SARSA is defined as following:
$C_{\gamma,\delta} = \max_{s_0,a_0}\sqrt{2\ln(12SA/\delta)}\cdot\sqrt{\sum_{h=0}^{\infty}\sum_{s,a}\gamma^{2h}\cdot\mu_h^{\hat{\pi}_\beta}(s,a|s_0,a_0)^2\,\mathrm{Var}\left[V^{\hat{\pi}_\beta}(s') \mid s,a\right]}$ *with* $\mu_h^\pi(s,a|s_0,a_0) := P^\pi(s_h = s, a_h = a, |s_0 = s, a_0 = a)$*, and $C_{CFPI}(s,a)$ denotes the error*

from the first-order approximation (3.1), (3.2) using CFPI, i.e.
$C_{\text{CFPI}}(s,a) := \left|(a-a_\beta)^T \left[\nabla_a \hat{Q}^{\hat{\pi}_\beta}(s,a)\right]_{a=a_\beta} + \hat{Q}^{\hat{\pi}_\beta}(s,a_\beta) - \hat{Q}^{\hat{\pi}_\beta}(s,a)\right|$. *Note that when* $a = a_\beta$, $C_{\text{CFPI}}(s,a) = 0$.

proof of Theorem 3.2.2. We focus on the first update, which is from $\hat{\pi}_b$ to $\hat{\pi}_1$. According to the Sarsa update, we have $|\hat{Q}^{\hat{\pi}_\beta}(s,a) - Q^{\hat{\pi}_\beta}(s,a)| \le \frac{C_{\gamma,\delta}}{\sqrt{\mathcal{D}(s,a)}}, \quad \forall s,a \in \mathcal{S}\times\mathcal{A}$ with probability $1-\delta$ and this is due to previous on-policy evaluation result (*e.g.* [66]). Also denote $\hat{\pi}_1 := \arg\max_\pi \bar{Q}^{\hat{\pi}_\beta}$. By Lemma 3.5.1,

$$
\begin{aligned}
J(\hat{\pi}_1) - J(\hat{\pi}_\beta) =& \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\hat{\pi}_1}}\left[\mathbb{E}_{a\sim\hat{\pi}_1(\cdot|s)} A^{\hat{\pi}_\beta}(s,a)\right] \\
=& \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\hat{\pi}_1}}\left[\mathbb{E}_{a\sim\hat{\pi}_1(\cdot|s)}[Q^{\hat{\pi}_\beta}(s,a) - V^{\hat{\pi}_\beta}(s)]\right] \\
=& \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\hat{\pi}_1}}\left[\mathbb{E}_{a\sim\hat{\pi}_1(\cdot|s)}[Q^{\hat{\pi}_\beta}(s,a) - Q^{\hat{\pi}_\beta}(s,\hat{\pi}_\beta(s))]\right] \\
\ge& \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\hat{\pi}_1}}\left[\mathbb{E}_{a\sim\hat{\pi}_1(\cdot|s)}[\hat{Q}^{\hat{\pi}_\beta}(s,a) - \hat{Q}^{\hat{\pi}_\beta}(s,\hat{\pi}_\beta(s))]\right] - \frac{2}{1-\gamma}\mathbb{E}_{s\sim d^{\hat{\pi}_1}}\mathbb{E}_{a\sim\hat{\pi}_1(\cdot|s)}\left[\frac{C_{\gamma,\delta}}{\sqrt{\mathcal{D}(s,a)}}\right] \\
\ge& \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\hat{\pi}_1}}\left[\bar{Q}^{\hat{\pi}_\beta}(s,\hat{\pi}_1(s)) - \bar{Q}^{\hat{\pi}_\beta}(s,\hat{\pi}_\beta(s))\right] - \frac{2}{1-\gamma}\mathbb{E}_{s\sim d^{\hat{\pi}_1}}\mathbb{E}_{a\sim\hat{\pi}_1(\cdot|s)}\left[\frac{C_{\gamma,\delta}}{\sqrt{\mathcal{D}(s,a)}} + C_{\text{CFPI}}(s,a)\right] \\
:=& \ \zeta^{(1)}.
\end{aligned}
$$

where the first inequality uses $|\hat{Q}^{\hat{\pi}_\beta}(s,a) - Q^{\hat{\pi}_\beta}(s,a)| \le \frac{C_{\gamma,\delta}}{\sqrt{\mathcal{D}(s,a)}}$ and the last inequality uses $\hat{\pi}_1 := \arg\max_\pi \bar{Q}^{\hat{\pi}_\beta}$. Here

$$
C_{\gamma,\delta} = \max_{s_0,a_0}\sqrt{2\ln(12SA/\delta)}\cdot\sqrt{\sum_{h=0}^{\infty}\sum_{s,a}\gamma^{2h}\cdot\mu_h^{\hat{\pi}_\beta}(s,a|s_0,a_0)\,\text{Var}\left[V^{\hat{\pi}_\beta}(s') \mid s,a\right]}
$$

Similarly, if the number of iteration $t > 1$, then Denote

$$
C_{\gamma,\delta}^{(t)} := \max_{s_0,a_0}\sqrt{2\ln(12SA/\delta)}\cdot\sqrt{\sum_{h=0}^{\infty}\sum_{s,a}\gamma^{2h}\cdot\frac{\mu_h^{\hat{\pi}_t}(s,a|s_0,a_0)^2}{\mu_h^{\hat{\pi}_{t-1}}(s,a|s_0,a_0)}\,\text{Var}\left[V^{\hat{\pi}_t}(s') \mid s,a\right]},
$$

then we have with probability $1-\delta$, by the Corollary 1 of [68], the OPE estimation follows

$$|\hat{Q}^{\hat{\pi}_\beta}(s,a) - Q^{\hat{\pi}_\beta}(s,a)| \leq \frac{C^{(t)}_{\gamma,\delta}}{\sqrt{\mathcal{D}(s,a)}}$$

and

$$\begin{aligned} J(\hat{\pi}_t) - J(\hat{\pi}_{t-1}) \geq & \frac{1}{1-\gamma}\mathbb{E}_{s\sim d^{\hat{\pi}_t}}\left[\bar{Q}^{\hat{\pi}_{t-1}}(s,\hat{\pi}_t(s)) - \bar{Q}^{\hat{\pi}_{t-1}}(s,\hat{\pi}_{t-1}(s))\right] \\ & - \frac{2}{1-\gamma}\mathbb{E}_{s\sim d^{\hat{\pi}_t}}\mathbb{E}_{a\sim\hat{\pi}_t(\cdot|s)}\left[\frac{C^{(t)}_{\gamma,\delta}}{\sqrt{\mathcal{D}(s,a)}} + C_{\mathrm{CFPI}}(s,a)\right] := \zeta^{(t)}, \end{aligned}$$

then for multi-step iterative algorithm, by a union bound, we have with probability $1-\delta$

$$J(\hat{\pi}_T) - J(\hat{\pi}_\beta) = \sum_{t=1}^{T} J(\hat{\pi}_t) - J(\hat{\pi}_{t-1}) \geq \sum_{t=1}^{T} \zeta^{(t)}.$$

□

On the learning coefficient of SARSA. The learning of SARSA is known to be statistically efficient from existing off-policy evaluation (OPE) literature, for instance [68, 69]. This is due to the on-policy SARSA scheme is just a special case of OPE task by choosing $\pi = \hat{\pi}_\beta$.

Concretely, we can translate the finite sample error bound in Corollary 1 of [68] to the infinite horizon discounted setting as: for any initial state,action s_0, a_0, with probability $1-\delta$,

$$\begin{aligned} |\hat{Q}^{\hat{\pi}_\beta}(s_0,a_0) - Q^{\hat{\pi}_\beta}(s_0,a_0)| \leq & \frac{1}{\sqrt{\mathcal{D}(s_0,a_0)}}\sqrt{2\ln(12/\delta)} \\ & \cdot \sqrt{\sum_{h=0}^{\infty}\sum_{s,a}\gamma^{2h}\cdot\mu_h^{\hat{\pi}_\beta}(s,a|s_0,a_0)\,\mathrm{Var}\left[V^{\hat{\pi}_\beta}(s') \mid s,a\right]} \end{aligned}$$

Note the original statement in [68] is for $v^{\hat{\pi}_\beta} - \hat{v}^{\hat{\pi}_\beta}$, here we conduct the version for $\hat{Q}^{\hat{\pi}_\beta} - Q^{\hat{\pi}_\beta}$ instead and this can be readily obtained by fixing the initial state action s_0, a_0 for v^π. As a result, by a union bound (over S, A) it is valid to define

$$C_{\gamma,\delta} = \max_{s_0,a_0} \sqrt{2\ln(12SA/\delta)} \cdot \sqrt{\sum_{h=0}^{\infty} \sum_{s,a} \gamma^{2h} \cdot \mu_h^{\hat{\pi}_\beta}(s,a|s_0,a_0) \operatorname{Var}\left[V^{\hat{\pi}_\beta}(s') \mid s,a\right]}$$

and this makes sure the statistical guarantee in Theorem 3.2.2 follows through.

Similarly, for the multi-step case, the OPE estimator hold with the corresponding coefficient

$$C_{\gamma,\delta}^{(t)} := \max_{s_0,a_0} \sqrt{2\ln(12SA/\delta)} \cdot \sqrt{\sum_{h=0}^{\infty} \sum_{s,a} \gamma^{2h} \cdot \frac{\mu_h^{\hat{\pi}_t}(s,a|s_0,a_0)^2}{\mu_h^{\hat{\pi}_{t-1}}(s,a|s_0,a_0)} \operatorname{Var}\left[V^{\hat{\pi}_t}(s') \mid s,a\right]}.$$

Lastly, even the assumption on the state-action space to be finite is not essential for Theorem 3.2.2 since, for more general function approximations, recent literature for OPE [70] shows SARSA update in Algorithm 1 is still statistically efficient.

3.6 Detailed Procedures to obtain Equation 3.13

We first highlight that we set the HP δ differently for Proposition 3.2.2 and 3.2.3. With the same τ, we generate the two different δ for the two different settings. Specifically,

$$\begin{aligned}\delta_{\text{lse}}(\tau) &= \log\tau + \min_i \left\{\frac{1}{2}\log\det(2\pi\Sigma_i) - \log\lambda_i\right\}, \quad \text{(Proposition 3.2.2)} \\ \delta_{\text{jensen}}(\tau) &= \log\tau + \frac{1}{2}\sum_{i=1}^{N}\lambda_i \log\det(2\pi\Sigma_i), \quad \text{(Proposition 3.2.3)}\end{aligned} \tag{3.46}$$

We next provide intuition for the design choices (3.46). Recall that the Gaussian Mixture behavior policy is constructed by

$$\pi_\beta = \sum_{i=1}^{N}\lambda_i \mathcal{N}(\mu_i, \Sigma_i). \tag{3.47}$$

With the mixture weights $\lambda_{i=1\dots N}$, we define the scaled probability $\breve{\pi}_i(a)$ of the i-th Gaussian component evaluated at a

$$\breve{\pi}_i(\mu_i) = \lambda_i \pi_i(a) = \lambda_i \det(2\pi\Sigma_i)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(a-\mu_i)^T\Sigma_i^{-1}(a-\mu_i)\}, \tag{3.48}$$

where $\pi_i(a) = \mathcal{N}(a;\mu_i,\Sigma_i)$ denotes the probability of the i-th Gaussian component evaluated at a. Therefore, we can have $\log\breve{\pi}_i(\mu_i) = \log\lambda_i - \frac{1}{2}\log\det(2\pi\Sigma_i)$, which implies that

$$\begin{aligned}\delta_{\text{lse}}(\tau) &= \log\tau + \min_i\left\{\frac{1}{2}\log\det(2\pi\Sigma_i) - \log\lambda_i\right\} \\ &= -\left(\max_i\left\{\log\lambda_i - \frac{1}{2}\log\det(2\pi\Sigma_i)\right\} - \log\tau\right). \\ &= -\max_i\left\{\log\frac{1}{\tau}\breve{\pi}_i(\mu_i)\right\}.\end{aligned} \tag{3.49}$$

By setting $\delta_{\text{lse}}(\tau)$ in this way, $\overline{\mu}_j = \overline{\mu}_j(\delta_{\text{lse}}(\tau))$ will satisfy the following condition whenever $\overline{\mu}_j$ is a valid solution to the sub-problem j (3.27) due to the KKT conditions, $\forall j \in$

$\{1, \ldots, N\}$.

$$\begin{aligned} & -\frac{1}{2}(\overline{\mu}_j - \mu_j)^T \Sigma_j^{-1}(\overline{\mu}_j - \mu_j) - \frac{1}{2}\log\det(2\pi\Sigma_j) + \log\lambda_j = -\delta_{\text{lse}}(\tau) \\ \iff \quad & \log \breve{\pi}_j(\overline{\mu}_j) = \max_i \left\{ \log \frac{1}{\tau} \breve{\pi}_i(\mu_i) \right\} \quad \iff \quad \breve{\pi}_j(\overline{\mu}_j) = \frac{1}{\tau} \max_i \{\breve{\pi}_i(\mu_i)\} \end{aligned} \tag{3.50}$$

To elaborate the design of $\delta_{\text{jensen}}(\tau)$, we first recall that the constraint of problem (3.11) is given by

$$\sum_{i=1}^{N} \lambda_i \left(-\frac{1}{2}\log\det(2\pi\Sigma_i) - \frac{1}{2}(\mu - \mu_i)^T \Sigma_i^{-1}(\mu - \mu_i) \right) \geq -\delta_{\text{jensen}}(\tau). \tag{3.51}$$

Note that the LHS of (3.51) is a concave function w.r.t μ. Thus, we can obtain its maximum by setting its derivatives (3.52) to zero

$$\begin{aligned} & \nabla_\mu \left(\sum_{i=1}^{N} \lambda_i \left(-\frac{1}{2}\log\det(2\pi\Sigma_i) - \frac{1}{2}(\mu - \mu_i)^T \Sigma_i^{-1}(\mu - \mu_i) \right) \right) \\ = & -\sum_{i=1}^{N} \lambda_i \Sigma_i^{-1}(\mu - \mu_i) = -\overline{\Sigma}^{-1}\mu + \overline{\Sigma}^{-1}\overline{\mu} \end{aligned} . \tag{3.52}$$

Interestingly, we can find that the solution is given by $\mu = \overline{\mu}$. Plugging $\mu = \overline{\mu}$ into the LHS of (3.51), we can obtain its maximum as below

$$\begin{aligned} & -\frac{1}{2}\sum_{i=1}^{N} \lambda_i \log\det(2\pi\Sigma_i) - \frac{1}{2}\sum_{i=1}^{N} \lambda_i (\overline{\mu} - \mu_i)^T \Sigma_i^{-1}(\overline{\mu} - \mu_i) \\ \leq & \sum_{i=1}^{N} \lambda_i \left(-\frac{1}{2}\log\det(2\pi\Sigma_i) \right) = \sum_{i=1}^{N} \lambda_i \log \pi_i(\mu_i) \end{aligned} \tag{3.53}$$

The inequality holds as the covariance matrix Σ_i is a positive semi-definite matrix for $i \in \{1 \dots N\}$. Therefore, our choice of $\delta_{\text{jensen}}(\tau)$ can be interpreted as

$$\delta_{\text{jensen}}(\tau) = \log \tau + \frac{1}{2} \sum_{i=1}^{N} \lambda_i \log \det(2\pi\Sigma_i) = -(\sum_{i=1}^{N} \lambda_i \log \pi_i(\mu_i) - \log \tau) \tag{3.54}$$

3.7 Multi-step and iterative algorithms

By setting $T > 0$, we can derive multi-step and iterative algorithms. Thanks to the tractability of our CFPI operators $\mathcal{I}_{\text{SG}}$ and $\mathcal{I}_{\text{MG}}$, we can always perform the policy improvement step in-closed form. Therefore, there is no significant gap between multi-step and iterative algorithms with our CFPI operators. One can differentiate our multi-step and iterative algorithms by whether an algorithm trains the policy evaluation step $\mathcal{E}(\hat{Q}_{t-1}, \hat{\pi}_t, \mathcal{D})$ to convergence or not.

As for the policy evaluation operator $\mathcal{E}$, the fitted Q evaluation [71, 72, 73] with a target network [74] has been demonstrated to be an effective and successful paradigm to perform policy evaluation [9, 10, 75, 76, 77] in deep (offline) RL. When instantiating a multi-step or iterative algorithm from Algorithm 1, one can also consider the other policy evaluation operators by incorporating more optimization techniques.

In the rest of this section, we will instantiate an iterative algorithm with our CFPI operators performing the policy improvement step and evaluate its effectiveness on the challenging AntMaze domains.

3.7.1 Iterative algorithm with our CFPI operators

In Sec. 3.4.1, we instantiate an iterative algorithm *Iterative* $\mathcal{I}_{MG}$ with our CFPI operator $\mathcal{I}_{\text{MG}}$. Algorithm 2 presents the corresponding pseudo-codes that learn a set of Q-function networks for simplicity. Without loss of generality, we can easily generalize the algorithm to learn the action-value distribution $Z(s, a)$ as is defined in (3.57).

For each task, we learn a Gaussian Mixture behavior policy $\hat{\pi}_\beta$ with behavior cloning. Similar to Sec. 3.4.1, we employed the IQN [58] architecture to model the Q-value network for its better generalizability. As our CFPI operator $\mathcal{I}_{\text{MG}}$ returns a deterministic policy, we follow the TD3 [77] to perform policy smoothing by adding noise to the action $a'(s')$

Algorithm 2 Iterative $\mathcal{I}_{\text{MG}}$

Input: Learned behavior policy $\hat{\pi}_\beta$, Q network parameters ϕ_1, ϕ_2, target Q network parameters $\phi_{\text{targ},1}$, $\phi_{\text{targ},2}$, dataset $\mathcal{D}$, parameter τ

1: **Repeat**
2: Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
3: Compute target actions:
$a'(s') = \text{clip}\left(\mathcal{I}_{\text{MG}}(\hat{\pi}_\beta, \hat{Q}; \tau)(s') + \text{clip}(\epsilon, -c, c), a_{Low}, a_{High}\right)$,
where $\hat{Q} = \min(Q_{\phi_1}, Q_{\phi_2})$, and $\epsilon \sim \mathcal{N}(0, \sigma)$
4: Compute targets: $y(r, s', d) = r + \gamma(1 - d) \min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', a'(s'))$
5: Update Q-functions by one step of gradient descent using
$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \sim B} \left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2$ for $i = 1, 2$
6: Update target networks with: $\phi_{\text{targ},i} \leftarrow \rho\phi_{\text{targ},i} + (1 - \rho)\phi_i$ for $i = 1, 2$
7: **Until** convergence
8: **Output**: $\mathcal{I}_{\text{MG}}(\hat{\pi}_\beta, \hat{Q}; \tau)$

in Line 4. After convergence, Algorithm 2 outputs an improved policy $\mathcal{I}_{\text{MG}}(\hat{\pi}_\beta, \hat{Q}; \tau)$.

Table 3.5 compares our Iterative $\mathcal{I}_{\text{MG}}$ with SOTA algorithms on the AntMaze domain. The performance for all baseline methods is directly reported from the IQL paper [52]. Our method outperforms all baseline methods on on 5 out of 6 tasks and obtaining the best overall performance. The training curves are shown in Fig. 3.2 with the HP settings detailed in Table 3.6. We did not perform much HP tuning, and thus one should expect a performance improvement after conducting fine-grained HP tuning.

Table 3.5: Comparison between our iterative algorithm and SOTA methods on the AntMaze domain of D4RL. We report the mean and standard deviation across 5 seeds for our methods. Our Iterative $\mathcal{I}_{\text{MG}}$ outperforms all baselines on 5 out of 6 tasks and obtaining the best overall performance, demonstrating the effectiveness of our CFPI operator when instantiating an iterative algorithm.

Dataset	BC	DT	Onestep RL	TD3+BC	CQL	IQL	Iterative $\mathcal{I}_{\text{MG}}$
antmaze-umaze-v0	54.6	59.2	64.3	78.6	74.0	87.5	$\mathbf{90.2 \pm 3.9}$
antmaze-umaze-diverse-v0	45.6	49.3	60.7	71.4	**84.0**	62.2	58.6 ± 15.2
antmaze-medium-play-v0	0.0	0.0	0.3	10.6	61.2	71.2	$\mathbf{75.2 \pm 6.9}$
antmaze-medium-diverse-v0	0.0	0.7	0.0	3.0	53.7	70.0	$\mathbf{72.2 \pm 7.3}$
antmaze-large-play-v0	0.0	0.0	0.0	0.2	15.8	39.6	$\mathbf{51.4 \pm 7.7}$
antmaze-large-diverse-v0	0.0	1.0	0.0	0.0	14.9	47.5	$\mathbf{52.4 \pm 10.9}$
Total	100.2	112.2	125.3	163.8	303.6	378.0	$\mathbf{400.0 \pm 52.0}$

3.8 CFPI beyond Gaussian policies

In this thesis, we mainly discuss the scenario when the behavior policy π_β is from the Gaussian family and develop two CFPI operators. However, our methods can also work with a non-Gaussian π_β. Next, we derive a new CFPI operator $\mathcal{I}_{\text{DET}}$ that can work with deterministic π_β. We then show that $\mathcal{I}_{\text{DET}}$ can also be leveraged to improve a general stochastic policy π_β without knowing its actual expression, as long as we can sample from it.

3.8.1 Deterministic behavior policy

When modeling both $\pi = \mu$ and $\pi_\beta = \mu_\beta$ as deterministic policies, we can derive the following BCPO from the problem (3.2) by setting $D(\cdot, \cdot)$ as the mean squared error.

$$\max_{\mu} \quad \mu^T \left[\nabla_a Q(s,a)\right]_{a=\mu_\beta}, \quad \text{s.t.} \quad \frac{1}{2}\|\mu - \mu_\beta\|^2 \leq \delta. \tag{3.55}$$

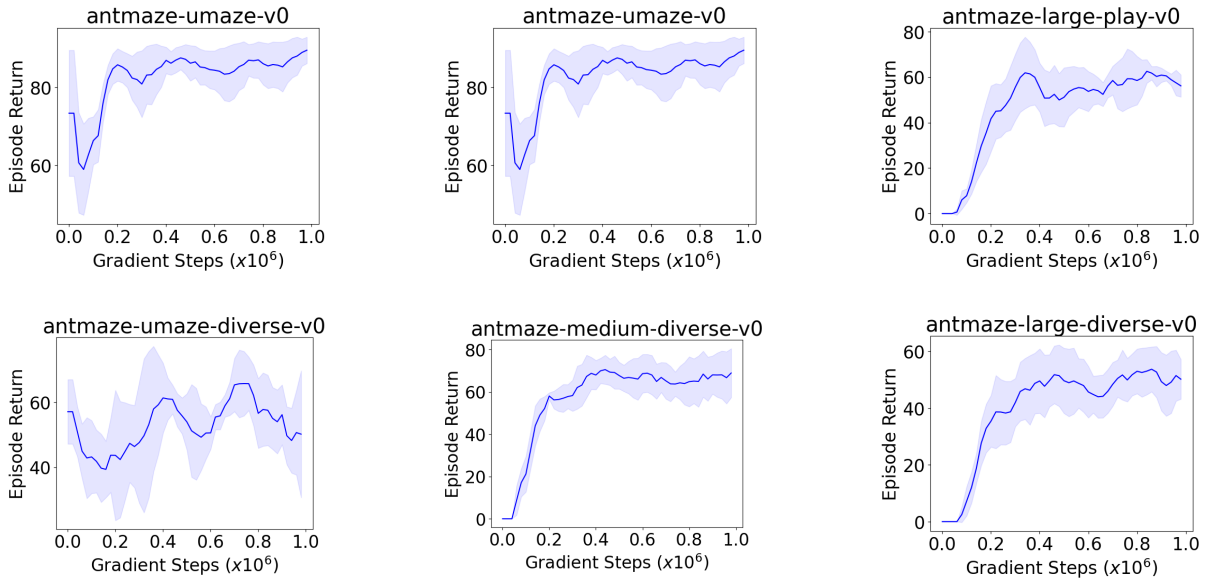


Figure 3.2: Iterative $\mathcal{I}_{\mathrm{MG}}$ training results on AntMaze. Shaded area denotes one standard deviation.

Algorithm 3 Policy improvement of $\mathcal{I}_{\mathrm{DET}}$ with a stochastic π_β

Input: State s, stochastic policy π_β, value function $\hat{Q}$, δ, number of candidate actions to sample M

1: Sample candidate actions $\{a_1, \ldots, a_M\}$ from π_β
2: Obtain the EBCQ policy π_{EBCQ} with action selected by $\pi_{\mathrm{EBCQ}}(s) = \arg\max_{m=1\ldots M} \hat{Q}(s, a_m)$
3: Return $\mathcal{I}_{\mathrm{DET}}(\pi_{\mathrm{EBCQ}}, \hat{Q}; \delta)$ by calculating (3.56)

Problem (3.55) has a similar form as the problem (3.17). We can thus obtain its closed-form solution $\mu = \mu_{\mathrm{det}}(\delta)$ as below

$$\mu_{\mathrm{det}}(\delta) = \mu_\beta + \frac{\sqrt{2\delta}}{\| \left[\nabla_a Q(s,a)\right]_{a=\mu_\beta} \|} \left[\nabla_a Q(s,a)\right]_{a=\mu_\beta}. \tag{3.56}$$

Therefore, we can derive a new CFPI operator $\mathcal{I}_{\mathrm{DET}}(\pi_\beta, Q; \delta)$ that returns a policy with action selected by (3.56).

We further note that the problem (3.55) can be seen as a linear approximation of the objectives used in TD3 + BC [10].

	Hyperparameter	Value
Shared HP	Optimizer	Adam [78]
	Normalize states	False
	activation function	ReLU
	Mini-batch size	256
MG-BC	Gaussian components (N)	8
	Number of gradient steps	500K
	Policy architecture	MLP
	Policy learning rate	1e-4
	Policy hidden layers	3
	Policy hidden dim	256
	Threshold ξ in (3.15)	0.05
Iterative $\mathcal{I}_{\text{MG}}$	Number of gradient steps	1M
	Critic architecture	IQN [58]
	Critic hidden dim	256
	Critic hidden layers	3
	Critic learning rate	3e-4
	Number of quantiles N_{q}	8
	Number of cosine basis elements	64
	Discount factor	0.99
	Target update rate	5e-3
	Target update period	1
	$\log \tau$	1.5

Table 3.6: Hyperparameters for our Iterative $\mathcal{I}_{\text{MG}}$.

3.8.2 Beyond deterministic behavior policy

Though we assume π_β to be a deterministic policy during the derivation of $\mathcal{I}_{\text{DET}}$, we can indeed leverage $\mathcal{I}_{\text{DET}}$ to tackle the more general case when we can only sample from π_β without knowing its actual expression.

Algorithm 3 details the procedures to perform the policy improvement step for a stochastic behavior policy π_β. We first obtain its EBCQ policy π_{EBCQ} in Line 1-2. As π_{EBCQ} is deterministic, we further plug it in $\mathcal{I}_{\text{DET}}$ in Line 3 to return an improved policy.

Table 3.7: $\mathcal{I}_{\text{DET}}$ results on the Gym-MuJoCo domain. We report the mean and standard deviation 5 seeds and each seed evaluates for 100 episodes.

Dataset	DET-BC	VAE-BC	VAE-EBCQ	$\mathcal{I}_{\text{DET}}$ with π_{det}	$\mathcal{I}_{\text{DET}}$ with π_{vae}
Walker2d-Medium-v2	71.2 ± 2.0	70.6 ± 3.0	70.6 ± 3.4	79.5 ± 12.9	$\mathbf{86.5 \pm 6.3}$
Walker2d-Medium-Replay-v2	19.5 ± 12.6	19.4 ± 2.9	33.5 ± 7.3	57.1 ± 11.6	$\mathbf{62.6 \pm 7.1}$
Walker2d-Medium-Expert-v2	74.4 ± 0.4	74.9 ± 7.6	82.7 ± 11.9	$\mathbf{111.2 \pm 1.8}$	$\mathbf{111.1 \pm 0.9}$

3.8.3 Experiment results

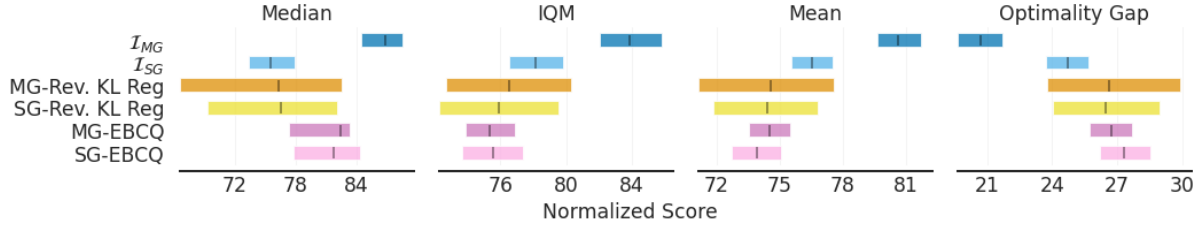
To evaluate the performance of $\mathcal{I}_{\text{DET}}$, we first learn two behavior policies with two different models. Specifically, we model π_{det} with a three-layer MLP that outputs a deterministic policy and π_{vae} with the Variational auto-encoder (VAE) [79] from BCQ [4]. Moreover, we reused the same value function $\hat{Q}_0$ as in Section 3.4.1. We present the results in Table 3.7. DET-BC and VAE denote the performance of π_{det} and π_{vae}, respectively. VAE-EBCQ denotes the EBCQ performance of π_{vae} with $M = 50$ candidate actions. Since π_{det} is deterministic, its EBCQ performance is the same as DET-BC. As for our two methods, we set $\delta = 0.1$ for all datasets. We can observe that both our $\mathcal{I}_{\text{DET}}$ with π_{det} and $\mathcal{I}_{\text{DET}}$ with π_{vae} largely improve over the baseline methods. Moreover, $\mathcal{I}_{\text{DET}}$ with π_{vae} outperforms VAE-EBCQ by a significant margins on all three datasets, demonstrating the effectiveness of our CFPI operator.

Indeed, our method benefits from an accurate and expressive behavior policy, as $\mathcal{I}_{\text{DET}}$ with π_{vae} achieves a higher average performance compared to $\mathcal{I}_{\text{DET}}$ with π_{det}, while maintaining a lower standard deviation on all three datasets.

We also note that we did not spend too much effort optimizing the HP, e.g., the VAE architectures, learning rates, and the value of τ.

3.9 Reliable evaluation to address the statistical uncertainty

Figure 3.3: Comparison between our methods and baselines using reliable evaluation methods proposed in [63]. We re-examine the results in Table 3.4 on the 9 tasks from the D4RL MuJoCo Gym domain. Each metric is calculated with a 95% CI bootstrap based on 9 tasks and 10 seeds for each task. Each seed further evaluates each method for 100 episodes. The interquartile mean (IQM) discards the top and bottom 25% data points and calculates the mean across the remaining 50% runs. The IQM is more robust as an estimator to outliers than the mean while maintaining less variance than the median. Higher median, IQM, mean scores, and lower Optimality Gap correspond to better performance. Our $\mathcal{I}_{\text{MG}}$ outperforms the baseline methods by a significant margin based on all four metrics.



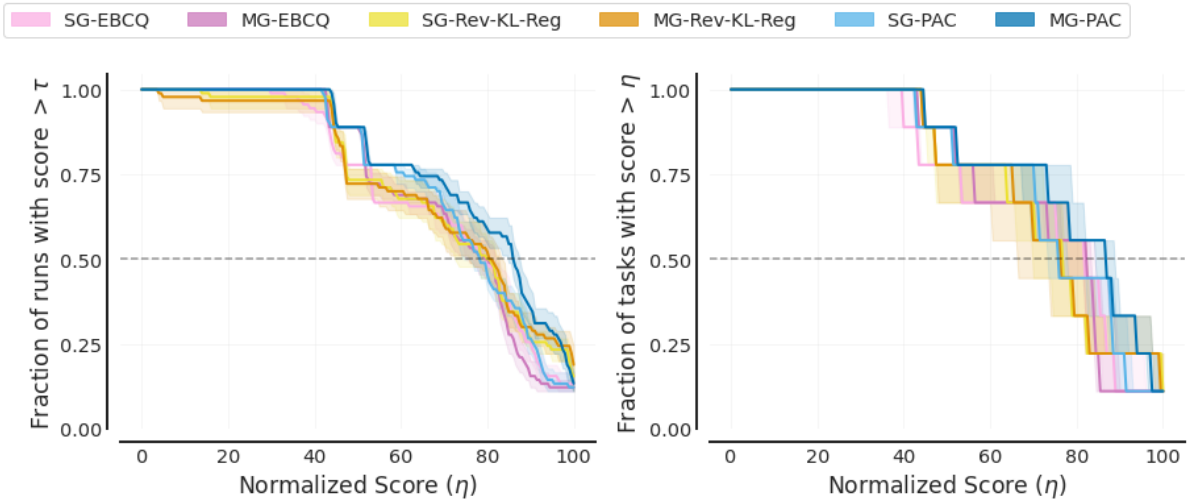
To demonstrate the superiority of our methods over the baselines and provide reliable evaluation results, we follow the evaluation protocols proposed in [63] to re-examine the results in Table 3.4. Specifically, we adopt the evaluation methods for all methods with $N_{\text{tasks}} \times N_{\text{seeds}}$ runs in total.

Moreover, we obtain the performance profile of each method, revealing its score distribution and variability. In particular, the score distribution shows the fraction of runs above a certain threshold η and is given by

$$\hat{F}(\eta) = \hat{F}\left(\eta; x_{1:N_{\text{tasks}},1:N_{\text{seeds}}}\right) = \frac{1}{N_{\text{tasks}}} \sum_{m=1}^{N_{\text{tasks}}} \frac{1}{N_{\text{seeds}}} \sum_{n=1}^{N_{\text{seeds}}} \mathbb{1}\left[x_{m,n} \geq \eta\right]$$

Evaluation results in Fig. 3.3 and Fig. 3.4 demonstrate that our $\mathcal{I}_{\text{MG}}$ outperforms the baseline methods by a significant margin based on all four reliable metrics.

Figure 3.4: Performance profiles (score distributions) for all methods on the 9 tasks from the D4RL MuJoCo Gym domain. The average score is calculated by averaging all runs within one task. Each task contains 10 seeds, and each seed evaluates for 100 episodes. Shaded area denotes 95% confidence bands based on percentile bootstrap and stratified sampling [63]. The η value where the curves intersect with the dashed horizontal line $y = 0.5$ corresponds to the median, while the area under the performance curves corresponds to the mean.



3.10 Hyper-parameter settings and training details

For all methods we proposed in Table 3.1, Table 3.3, and Table 3.4, we obtain the mean and standard deviation of each method across 10 seeds. Each seed contains individual training process and evaluates the policy for 100 episodes.

3.10.1 HP and training details for methods in Table 3.1 and Table 3.4

Table 3.8 includes the HP of methods evaluated on the Gym-MuJoCo domain. We use the Adam [78] optimizer for all learning algorithms and normalize the states in each dataset following the practice of TD3+BC [10]. Note that our one-step offline RL algorithms presented in Table 3.1 (Our $\mathcal{I}_{\text{MG}}$) and Table 3.4 ($\mathcal{I}_{\text{MG}}$, $\mathcal{I}_{\text{SG}}$, MG-EBCQ, SG-

EBCQ, MG-MS) require learning a behavior policy and the value function $\hat{Q}^0$. Therefore, we will first describe the detailed procedures for learning Single Gaussian (SG-BC) and Gaussian Mixture (MG-BC) behavior policies. We next describe our SARSA-style training procedures to estimate $\hat{Q}^0$. Finally, we will present the details for each one-step algorithm.

	Hyperparameter	Value
Shared HP	Optimizer	Adam [78]
	Normalize states	True
	Policy architecture	MLP
	Policy learning rate	1e-4
	Policy hidden layers	3
	Policy hidden dim	256
	Policy activation function	ReLU
	Threshold ξ in (3.15)	0.05
MG-BC	Gaussian components (N)	4
	Number of gradient steps	500K
	Mini-batch size	256
SG-BC	Number of gradient steps	500K
	Mini-batch size	512
SARSA	Number of gradient steps	Table 3.16
	Critic architecture	IQN [58]
	Critic hidden dim	256
	Critic hidden layers	3
	Critic activation function	ReLU
	Number of quantiles N_{q}	8
	Number of cosine basis elements	64
	Discount factor	0.99
	Target update rate	5e-3
	Target update period	1
Our $\mathcal{I}_{\text{MG}}$ (Table 3.1)	$\log \tau$	0 for Hopper-M-E; 0.5 for the others
$\mathcal{I}_{\text{MG}}$ & $\mathcal{I}_{\text{SG}}$(Table 3.4)	$\log \tau$	0.5 for all tasks
MG-EBCQ	Number of candidate actions N_{bcq}	5
SG-EBCQ	Number of candidate actions N_{bcq}	10
MG-Rev. KL Reg	α	3.0
& SG-Rev. KL Reg	Number of gradient steps	100K

Table 3.8: Hyperparameters for our methods in Table 3.1 and Table 3.4.

For the $\mathcal{I}_{\text{MG}}$ results on the D4RL Mujoco dataset, we sample 10 τ from a range of

[0.5, 1.0] or [0, 0.1]. We found that our method worked robustly for these ranges. For the $\mathcal{I}_{\text{SG}}$, we also sample 10 τ just from a range of [0.5, 1.0]. We use a target quantile of 0.7, and scale all τ accordingly to 3.6. For behavior cloning, we use

MG-BC. We parameterize the policy as a 3-layer MLP, which outputs the tanh of a Gaussian Mixture with $N = 4$ Gaussian components. For each Gaussian component, we learn the state-dependent diagonal covariance matrix. While existing methods suggest learning Gaussian Mixture via expectation maximization [80, 81, 82] or variational Bayes [83], we empirically find that directly minimizing the negative log-likelihood of actions sampled from the offline datasets achieves satisfactory performance, as is shown in Table 3.1. We train the policy for 500K gradient steps. We emphasize that we do not aim to propose a better algorithm for learning a Gaussian Mixture behavior policy. Instead, future work may use a more advanced algorithm to capture the underlying behavior policy better.

SG-BC. We parameterize the policy as a 3-layer MLP, which outputs the tanh of a Single Gaussian with the state-dependent diagonal covariance matrix [45, 75]. We train the policy for 500K gradient steps.

SARSA. We parameterize the value function with the IQN [58] architecture and train it to model the distribution $Z^\beta : \mathcal{S} \times \mathcal{A} \to \mathcal{Z}$ of the behavior return via quantile regression, where $\mathcal{Z}$ is the action-value distributional space [84] defined as

$$\mathcal{Z} = \{Z : \mathcal{S} \times \mathcal{A} \to \mathscr{P}(\mathbb{R}) \mid \mathbb{E}\left[|Z(s,a)|^p\right] < \infty, \forall (s,a), p \geq 1\}. \tag{3.57}$$

We define the CDF function of Z^β as $F_{Z^\beta}(z) = Pr(Z^\beta < z)$, leading to the quantile function [85] $F^{-1}_{Z^\beta}(\rho) := \inf\{z \in \mathbb{R} : \rho \leq F_{Z^\beta}(z)\}$ as the inverse CDF function, where ρ denotes the quantile fraction. We further denote $Z^\beta_\rho = F^{-1}_{Z^\beta}(\rho)$ to ease the notation.

To obtain Z^β, we leverage the empirical distributional bellman operator $\hat{\mathcal{T}}^\beta_D : \mathcal{Z} \to \mathcal{Z}$

defined as

$$\hat{\mathcal{T}}_D^{\beta} Z(s,a) :\overset{D}{=} r + \gamma Z\left(s', a'\right) \mid (s,a,r,s',a') \sim \mathcal{D}, \tag{3.58}$$

where $A :\overset{D}{=} B$ implies the random variables A and B are governed by the same distribution. We note that $\hat{\mathcal{T}}_D^{\beta}$ helps to construct a Huber quantile regression loss [58, 84, 86], and we can finally learn Z^{β} by minimizing the quantile regression loss following a similar procedures as in [84].

To achieve the goal, we approximate Z^{β} by N_{q} quantile fractions $\{\rho_i \in [0,1] \mid i = 0 \dots N_{\mathrm{q}}\}$ with $\rho_0 = 0$, $\rho_{N_{\mathrm{q}}} = 1$ and $\rho_i < \rho_j, \forall i < j$. We further denote $\hat{\rho}_i = (\rho_i + \rho_{i+1})/2$, and use random sampling [58] to generate the quantile fractions. By further parameterizing $Z_{\rho}^{\beta}(s,a)$ as $\hat{Z}_{\rho}^{\beta}(s,a;\theta)$ with parameter θ, we can derive the loss function $J_Z(\theta)$ as

$$J_Z(\theta) = \mathbb{E}_{(s,a,r,s',a') \sim \mathcal{D}} \left[\sum_{i=0}^{N_{\mathrm{q}}-1} \sum_{j=0}^{N_{\mathrm{q}}-1} \left(\rho_{i+1} - \rho_i\right) l_{\hat{\rho}_j}\left(\delta_{ij}\right) \right],$$

$$\text{where} \quad \delta_{ij} = \delta_{ij}\left(s,a,r,s',a'\right) = r + \gamma Z_{\hat{\rho}_i}\left(s', a'; \bar{\theta}\right) - Z_{\hat{\rho}_j}\left(s,a;\theta\right)$$

$$\text{and} \quad l_{\rho}\left(\delta_{ij}\right) = \left|\rho - \mathbb{I}\left\{\delta_{ij} < 0\right\}\right| \mathcal{L}\left(\delta_{ij}\right), \text{ with } \mathcal{L}\left(\delta_{ij}\right) = \begin{cases} \frac{1}{2}\delta_{ij}^2, & \text{if } |\delta_{ij}| \leq 1 \\ |\delta_{ij}| - \frac{1}{2}, & \text{otherwise.} \end{cases} \tag{3.59}$$

$\bar{\theta}$ is the parameter of the target network [76] given by the Polyak averaging of θ. We refer interested readers to [58, 84] for further details.

The training procedures above returns $\hat{Z}_{\rho}^{\beta}, \forall \rho \in [0,1]$. With the learned $\hat{Z}_{\rho}^{\beta}$, our one-step methods presented in Table 3.1 and Table 3.4 extract the value function by setting $\hat{Q}_0 = \mathbb{E}_{\rho}[\hat{Z}_{\rho}^{\beta}] = \hat{Q}^{\beta}$ as the expectation of $\hat{Z}_{\rho}^{\beta}$, which is equivalent to the conventional action-value function $\hat{Q}^{\beta}$. Specifically, we use $N = 32$ fixed quantile fractions with

$\rho_i = i/N,\ i = 0 \dots N$. Given a state-action pair (s, a), we calculate $\hat{Q}_0(s, a) = \hat{Q}^\beta(s, a)$ as

$$\hat{Q}_0(s, a) = \hat{Q}^\beta(s, a) = \frac{1}{N} \sum_{i=1}^{N} \hat{Z}^\beta_{\hat{\rho}_i}(s, a), \quad \hat{\rho}_i = \frac{\rho_i + \rho_{i-1}}{2}. \tag{3.60}$$

Since our methods still need to query out-of-buffer action values during rollout, we employed the conventional double Q-learning [77] technique to prevent potential overestimation without clipping. Specifically, we initialize $\hat{Q}^1_0$ and $\hat{Q}^2_0$ differently and train them to minimize (3.59). With the learned $\hat{Q}^1_0$ and $\hat{Q}^2_0$, we set the value of $\hat{Q}_0(s, a)$ as

$$\hat{Q}_0(s, a) = \min_{k=1,2} \hat{Q}^k_0(s, a) \tag{3.61}$$

for every (s, a) pair. Note that the double Q-learning technique is only used during policy evaluation.

As for deciding the number of gradient steps, we detail our procedures in subsection 3.11.5. And the number of gradient steps for each dataset can be found in Table 3.16.

Our $\mathcal{I}_{\mathrm{MG}}$ (Table 3.1). Recall that our CFPI operator $\mathcal{I}_{\mathrm{MG}}(\hat{\pi}_\beta, \hat{Q}_0; \tau)$ requires to learn a Gaussian Mixture behavior policy $\hat{\pi}_\beta$ and a value function $\hat{Q}_0$. We train $\hat{\pi}_\beta$ and $\hat{Q}_0$ according to the procedures listed in **MG-BC** and **SARSA**, respectively. By following the practice of [12, 45], we perform a grid search on $\log \tau \in \{0, 0.5, 1.0, 1.5, 2.0\}$ using 3 seeds. We note that we manually reduce $\mathcal{I}_{\mathrm{MG}}$ to MG-MS when $\log \tau = 0$ by only considering the mean of each non-trivial Gaussian component. Our results show that setting $\log \tau = 0.5$ achieves the best overall performance while Hopper-M-E requires an extremely small $\log \tau$ to perform well as is shown in subsection 3.11.2. Therefore, we decide to set $\log \tau = 0$ for Hopper-M-E and $\log \tau = 0.5$ for the other 8 datasets. We then obtain the results for the other 7 seeds with these HP settings and report the results on the 10 seeds in total.

Table 3.9: HP search for MG-EBCQ. We report the mean and std of 10 seeds, and each seed evaluates for 100 episodes.

Dataset	$N_{\text{bcq}} = 2$	$N_{\text{bcq}} = 5$	$N_{\text{bcq}} = 10$	$N_{\text{bcq}} = 20$	$N_{\text{bcq}} = 50$	$N_{\text{bcq}} = 100$
Cheetah-M-v2	47.2 ± 0.3	51.5 ± 0.2	53.3 ± 0.3	54.4 ± 0.3	55.3 ± 0.4	55.8 ± 0.4
Hopper-M-v2	63.3 ± 2.3	82.5 ± 1.9	88.3 ± 4.6	90.8 ± 6.9	92.1 ± 7.6	91.3 ± 9.4
Walker2d-M-v2	78.6 ± 1.4	85.2 ± 2.1	81.0 ± 6.3	73.4 ± 11.0	67.6 ± 14.8	62.7 ± 15.8
Cheetah-M-R-v2	38.8 ± 0.6	43.0 ± 0.3	44.2 ± 0.3	44.7 ± 0.5	44.8 ± 0.8	44.6 ± 0.8
Hopper-M-R-v2	58.4 ± 6.9	83.6 ± 10.3	82.8 ± 14.9	82.3 ± 15.9	77.5 ± 17.7	76.0 ± 16.7
Walker2d-M-R-v2	55.1 ± 3.4	73.1 ± 5.2	75.6 ± 5.3	77.6 ± 5.4	78.0 ± 5.6	78.5 ± 4.5
Cheetah-M-E-v2	75.2 ± 3.2	84.5 ± 4.6	82.7 ± 5.2	77.6 ± 7.3	73.4 ± 6.4	68.8 ± 5.9
Hopper-M-E-v2	73.6 ± 7.5	56.1 ± 6.2	44.9 ± 4.6	37.3 ± 3.6	29.8 ± 2.9	25.3 ± 3.3
Walker2d-M-E-v2	107.1 ± 1.8	111.1 ± 1.0	111.4 ± 1.5	111.4 ± 2.5	109.6 ± 4.0	107.2 ± 6.0
Total	597.2 ± 27.4	670.6 ± 31.9	664.1 ± 43.1	649.5 ± 53.5	628.0 ± 60.2	610.2 ± 62.9

$\mathcal{I}_{\text{MG}}$ (Table 3.4) & $\mathcal{I}_{\text{SG}}$ (Table 3.4). Different from the results in Table 3.1, we use the same $\log \tau = 0.5$ for all datasets including Hopper-M-E to obtain the performance of $\mathcal{I}_{\text{MG}}$ in Table 3.4. In this way, we aim to understand the effectiveness of each component of our methods better. To fairly compare $\mathcal{I}_{\text{MG}}$ and $\mathcal{I}_{\text{SG}}$, we tune the τ for $\mathcal{I}_{\text{SG}}$ in a similar way by performing a grid search on $\log \tau \in \{0.5, 1.0, 1.5, 2.0\}$ with 3 seeds and finally set $\log \tau = 0.5$ for all datasets. We then obtain the results for the other 7 seeds and report the results with 10 seeds in total.

MG-EBCQ & SG-EBCQ. We tune the number of candidate actions N_{bcq} from the same range $\{2, 5, 10, 20, 50, 100\}$ as is in [12]. For each N_{bcq}, we obtain its average performance for all tasks across 10 seeds and select the best performing N_{bcq} for each method. We separately tune the N_{bcq} for MG-EBCQ and SG-EBCQ. As a result, we set $N_{\text{bcq}} = 5$ for MG-EBCQ and $N_{\text{bcq}} = 10$ for SG-EBCQ. **Moreover, we highlight that MG-EBCQ (SG-EBCQ) uses the same behavior policy and value function as is in $\mathcal{I}_{\text{MG}}$ ($\mathcal{I}_{\text{SG}}$).** We include the full hyper-parameter search results in Table 3.9 and Table 3.10.

MG-Rev. KL Reg & SG-Rev. KL Reg. We tune the regularization strength α from the same range $\{0.03, 0.1, 0.3, 1.0, 3.0, 10.0\}$ as is in [12]. For each α, we obtain

Table 3.10: HP search for SG-EBCQ. We report the mean and std of 10 seeds, and each seed evaluates for 100 episodes.

Dataset	$N_{\text{bcq}} = 2$	$N_{\text{bcq}} = 5$	$N_{\text{bcq}} = 10$	$N_{\text{bcq}} = 20$	$N_{\text{bcq}} = 50$	$N_{\text{bcq}} = 100$
Cheetah-M-v2	47.1 ± 0.2	51.5 ± 0.1	53.3 ± 0.2	54.4 ± 0.3	55.3 ± 0.3	55.8 ± 0.4
Hopper-M-v2	60.7 ± 2.4	78.6 ± 4.0	86.8 ± 5.2	89.1 ± 7.7	89.8 ± 8.8	89.8 ± 9.8
Walker2d-M-v2	78.5 ± 2.8	86.9 ± 1.8	85.2 ± 5.1	81.5 ± 9.3	76.6 ± 11.8	72.4 ± 13.8
Cheetah-M-R-v2	37.8 ± 0.7	42.3 ± 0.6	43.5 ± 0.6	44.3 ± 0.7	44.1 ± 1.1	43.6 ± 0.9
Hopper-M-R-v2	58.7 ± 5.8	85.2 ± 9.0	88.5 ± 12.2	89.1 ± 11.7	83.9 ± 15.0	82.1 ± 16.1
Walker2d-M-R-v2	54.0 ± 7.2	72.2 ± 5.2	75.4 ± 4.6	77.7 ± 4.8	77.5 ± 5.8	74.9 ± 6.2
Cheetah-M-E-v2	71.8 ± 2.2	81.9 ± 4.8	81.8 ± 5.4	77.6 ± 6.9	71.5 ± 7.5	68.2 ± 6.5
Hopper-M-E-v2	66.4 ± 4.8	49.8 ± 6.2	40.0 ± 5.8	34.9 ± 6.2	29.0 ± 5.7	25.2 ± 4.8
Walker2d-M-E-v2	106.6 ± 1.6	111.0 ± 0.9	111.1 ± 1.8	110.0 ± 3.7	107.2 ± 7.8	106.0 ± 9.0
Total	581.6 ± 27.7	659.4 ± 32.7	665.5 ± 41.0	658.7 ± 51.3	634.7 ± 63.9	618.1 ± 67.5

Table 3.11: HP search for MG-Rev. KL Reg. We report the mean and std of 10 seeds, and each seed evaluates for 100 episodes.

Dataset	$\alpha = 0.03$	$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 1.0$	$\alpha = 3.0$	$\alpha = 10.0$
Cheetah-M-v2	58.3 ± 1.1	58.1 ± 1.2	55.6 ± 0.5	50.6 ± 0.3	47.0 ± 0.2	44.5 ± 0.2
Hopper-M-v2	14.4 ± 13.6	41.0 ± 31.0	89.4 ± 22.2	99.7 ± 1.1	76.3 ± 6.9	58.5 ± 4.0
Walker2d-M-v2	5.8 ± 4.8	18.4 ± 21.9	34.2 ± 27.3	82.2 ± 7.8	82.8 ± 1.8	76.9 ± 2.0
Cheetah-M-R-v2	46.7 ± 1.8	47.5 ± 1.6	48.1 ± 0.7	46.4 ± 0.6	44.4 ± 0.5	43.1 ± 0.4
Hopper-M-R-v2	70.9 ± 33.8	86.6 ± 26.3	103.1 ± 0.8	101.4 ± 1.1	99.4 ± 2.1	77.6 ± 17.2
Walker2d-M-R-v2	73.7 ± 28.8	65.4 ± 33.8	64.0 ± 39.9	65.4 ± 35.8	69.7 ± 30.9	57.7 ± 22.8
Cheetah-M-E-v2	0.4 ± 2.2	1.2 ± 1.9	4.0 ± 1.9	25.0 ± 6.3	65.0 ± 10.1	86.2 ± 7.1
Hopper-M-E-v2	2.6 ± 1.7	16.2 ± 7.9	22.5 ± 10.7	57.4 ± 23.6	79.4 ± 32.6	86.8 ± 15.7
Walker2d-M-E-v2	10.4 ± 15.3	25.5 ± 38.1	93.5 ± 34.5	109.8 ± 0.6	107.1 ± 4.0	97.4 ± 7.0
Total	283.2 ± 103.0	359.9 ± 163.5	514.3 ± 138.5	637.8 ± 77.2	671.2 ± 89.1	628.6 ± 76.4

its average performance for all tasks across 10 seeds and select the best performing α for each method. We separately tune the α for MG-Rev. KL Reg & SG-Rev. KL Reg, although $\alpha = 3.0$ achieves the best overall performance in both methods. **Moreover, we highlight that MG-Rev. KL Reg (SG-Rev. KL Reg) uses the same behavior policy and value function as is in $\mathcal{I}_{\mathbf{MG}}$ ($\mathcal{I}_{\mathbf{SG}}$).** We include the full hyper-parameter search results in Table 3.11 and Table 3.12.

Table 3.12: HP search for SG-Rev. KL Reg. We report the mean and std of 10 seeds, and each seed evaluates for 100 episodes.

Dataset	$\alpha = 0.03$	$\alpha = 0.1$	$\alpha = 0.3$	$\alpha = 1.0$	$\alpha = 3.0$	$\alpha = 10.0$
Cheetah-M-v2	58.6 ± 1.3	57.9 ± 0.8	55.2 ± 0.5	50.7 ± 0.5	47.1 ± 0.2	44.5 ± 0.3
Hopper-M-v2	18.7 ± 15.6	40.2 ± 24.7	83.2 ± 19.6	98.8 ± 2.0	70.3 ± 7.0	57.2 ± 4.6
Walker2d-M-v2	5.6 ± 3.5	26.2 ± 27.1	37.0 ± 27.6	83.3 ± 7.5	82.4 ± 1.0	77.1 ± 1.2
Cheetah-M-R-v2	46.1 ± 3.6	47.8 ± 1.3	47.8 ± 0.8	46.0 ± 0.5	44.3 ± 0.4	42.5 ± 0.6
Hopper-M-R-v2	77.4 ± 19.1	60.8 ± 27.7	92.0 ± 21.9	100.7 ± 1.0	99.7 ± 1.0	70.3 ± 19.2
Walker2d-M-R-v2	59.5 ± 31.3	72.7 ± 38.8	75.7 ± 30.4	75.1 ± 25.3	63.6 ± 28.5	59.7 ± 21.5
Cheetah-M-E-v2	1.1 ± 3.2	3.4 ± 3.4	7.1 ± 4.3	38.9 ± 18.4	78.9 ± 9.8	89.1 ± 4.0
Hopper-M-E-v2	5.5 ± 4.0	20.1 ± 8.6	24.8 ± 7.9	43.8 ± 23.6	76.6 ± 18.3	67.7 ± 30.6
Walker2d-M-E-v2	1.7 ± 3.7	13.4 ± 33.5	83.2 ± 37.5	109.9 ± 0.7	106.7 ± 4.1	96.8 ± 7.6
Total	274.0 ± 85.3	342.6 ± 165.9	505.9 ± 150.5	647.2 ± 79.5	669.7 ± 70.3	604.9 ± 89.5

3.10.2 HP and training details for methods in Table 3.3

Table 3.13 includes the HP for experiments in Sec. 3.4.2. The of IQL. We use the same HP for the IQL training as is reported in the IQL paper. We obtain the IQL policy π_{IQL} and Q_{IQL} by training for 1M gradient steps using the PyTorch Implementation from RLkit [62], a widely used RL library. We emphasize that we follow the authors' exact training and evaluation protocol. We include the training curves for all tasks from the AntMaze domain in subsection 3.11.6.

Note that IQL [52] reported inconsistent offline experiment results on AntMaze in its paper's Table 1, Table 2, Table 5, and Table 6 [2]. We suspect that these results are obtained from different sets of random seeds. In subsection 3.11.6, we present all these results in Table 3.17.

To obtain the performance for $\mathcal{I}_{\text{SG}}(\pi_{\text{IQL}}, Q_{\text{IQL}})$, we follow the practice of [12, 45] and perform a grid search on $\log \tau \in \{0.1, 0.2, 2.0\}$ using 3 seeds for each dataset. We then evaluate the best choice for each dataset by obtaining corresponding results on the other 7 seeds. We finally report the results with 10 seeds in total.

[2] Link to the IQL paper. IQL's Table 5 & 6 are presented in the supplementary material.

	Hyperparameter	Value
Shared HP	Normalize states	False
IQL HP	Optimizer	Adam [78]
	Number of gradient steps	1M
	Mini-batch size	256
	Policy learning rate	3e-4
	Policy hidden dim	256
	Policy hidden layers	2
	Policy activation function	ReLU
	Critic architecture	MLP
	Critic learning rate	3e-4
	Critic hidden dim	256
	Critic hidden layers	2
	Critic activation function	ReLU
	Target update rate	5e-3
	Target update period	1
	quantile	0.9
	temperature	10.0
$\mathcal{I}_{\mathrm{SG}}(\pi_{\mathrm{IQL}}, Q_{\mathrm{IQL}})$	$\log \tau$	selected from $\{0.1, 0.2, 2.0\}$

Table 3.13: Hyperparameters for methods in Table 3.3

3.11 Additional Experiments

3.11.1 Complete experiment results for MG-MS

Table 3.14 provides the results of MG-MS on the 9 tasks from the MuJoCo Gym domain in compensation for the results in Sec. 3.4.3.

Table 3.14: Results of MG-MS on the MuJoCo Gym domain. We report the mean and standard deviation across 10 seeds, and each seed evaluates for 100 episodes.

Dataset	MG-MS (3.15)
Cheetah-M-v2	43.6 ± 0.2
Hopper-M-v2	55.3 ± 6.3
Walker2d-M-v2	73.6 ± 2.2
Cheetah-M-R-v2	42.4 ± 0.4
Hopper-M-R-v2	61.5 ± 15.1
Walker2d-M-R-v2	65.0 ± 10.4
Cheetah-M-E-v2	91.3 ± 2.1
Hopper-M-E-v2	104.2 ± 5.1
Walker2d-M-E-v2	104.1 ± 6.7
Total	641.1 ± 48.5

3.11.2 Complete experiment results on the effect of the HP τ

Fig. 3.5 presents additional results in compensation for the results in Sec. 3.4.3. We note that Hopper-Medium-Expert-v2 requires a much smaller $\log \tau$ than the other tasks to perform well.

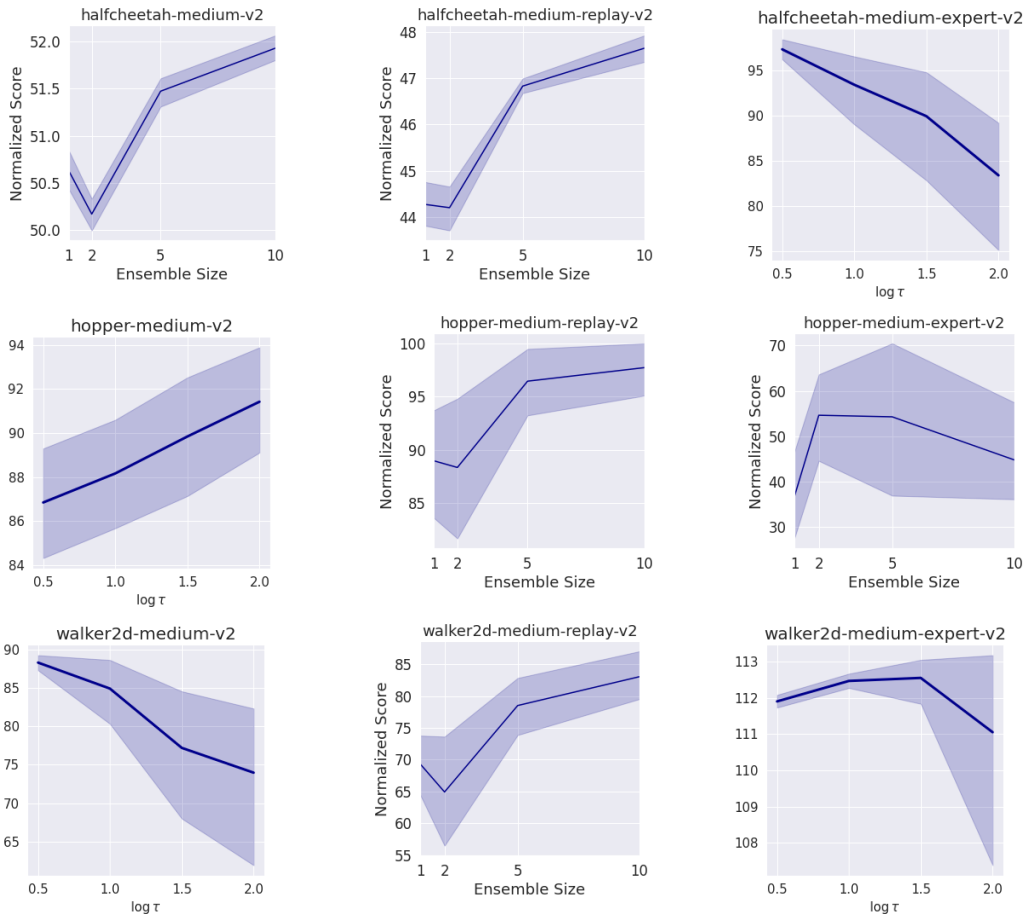


Figure 3.5: Performance of $\mathcal{I}_{\text{MG}}$ with varying $\log \tau$. The other HP can be found in Table 3.8. Each variant averages returns over 10 seeds, and each seed contains 100 evaluation episodes. The shaded area denotes bootstrapped 95% CI.

3.11.3 Ablation study on the number Gaussian components

In this section, we explore whether increasing the number of Gaussian components will result in a performance boost. We use the same settings as in Table 3.1 except modeling $\hat{\pi}_\beta$ with 8 Gaussian instead of 4. We hypothesize the performance gain should most likely happen on the three Medium-Replay datasets, as these datasets are collected by diverse policies. However, Table 3.15 shows that simply increasing the number of Gaussian components from 4 to 8 hardly results in a performance boost, as increasing the number of Gaussian components will induce extra optimization difficulties during

behavior cloning [82].

Table 3.15: Comparison between setting the number of Gaussian components to 4 and 8 for our $\mathcal{I}_{\text{MG}}$ on the three Medium-Replay datasets. We report the mean and standard deviation across 10 seeds, and each seed evaluates for 100 episodes.

Dataset	4 components (Table 3.1)	8 components
Cheetah-M-R-v2	44.5 ± 0.4	44.3 ± 0.3
Hopper-M-R-v2	93.6 ± 7.9	90.6 ± 11.6
Walker2d-M-R-v2	78.2 ± 5.6	79.4 ± 4.5

3.11.4 Modeling the value network with conventional MLP

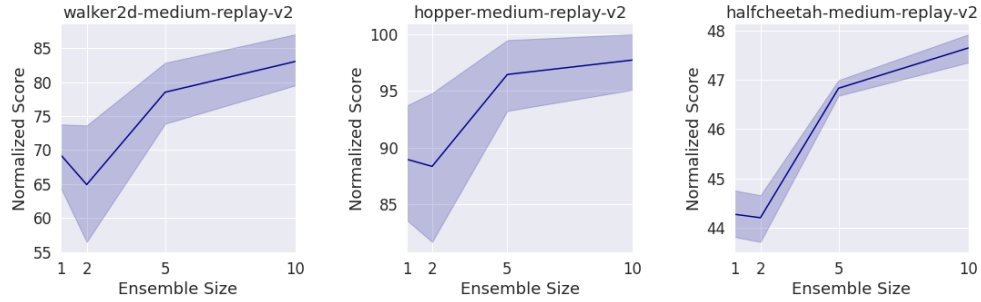


Figure 3.6: Performance of $\mathcal{I}_{\text{MG}}$ with varying ensemble sizes. Each variant averages returns over 8 seeds, and each seed contains 100 evaluation episode. Each Q-value network is modeled by a 3-layer MLP. The shaded area denotes bootstrapped 95% CI.

Our experiments in Sec. 3.4.1 rely on learning a Q value function with the IQN [58] architecture. In this section, we examine the effectiveness of our CFPI operator $\mathcal{I}_{\text{MG}}$ when working with an ensemble of conventional MLP Q-value networks with varying ensemble sizes M.

Each Q-value network $\hat{Q}^{\text{MLP}}_{\theta_k}$ uses ReLU activation and is parameterized with θ_k, including 3 hidden layers of width 256. We train each $\hat{Q}^{\text{MLP}}_{\theta_k}$ by minimizing the bellman error below

$$L(\theta_k) = \mathbb{E}_{(s,a,r,s',a')\sim\mathcal{D}}\left[r + \gamma \hat{Q}^{\text{MLP}}(s', a'; \bar{\theta_k}) - \hat{Q}^{\text{MLP}}(s, a; \theta_k)\right], \tag{3.62}$$

where $\bar{\theta_k}$ is the parameter of a target network given by the Polyak averaging of θ. We set $\hat{Q}^{\text{MLP}}(s, a; \theta_k) = \hat{Q}^{\text{MLP}}_{\theta_k}(s, a)$. We further note that Equation 3.61 can be reformulated as

$$\begin{aligned}\hat{Q}_0(s,a) = \min_{k=1,2} \hat{Q}^k_0(s,a) &= \frac{1}{2}|\hat{Q}^1_0(s,a) + \hat{Q}^2_0(s,a)| - \frac{1}{2}|\hat{Q}^1_0(s,a) - \hat{Q}^2_0(s,a)| \\ &= \hat{\mu}_Q(s,a) - \hat{\sigma}_Q(s,a),\end{aligned} \tag{3.63}$$

where $\hat{\mu}_Q$ and $\hat{\sigma}_Q$ calculate the mean and standard deviation of Q value [51]. In the case

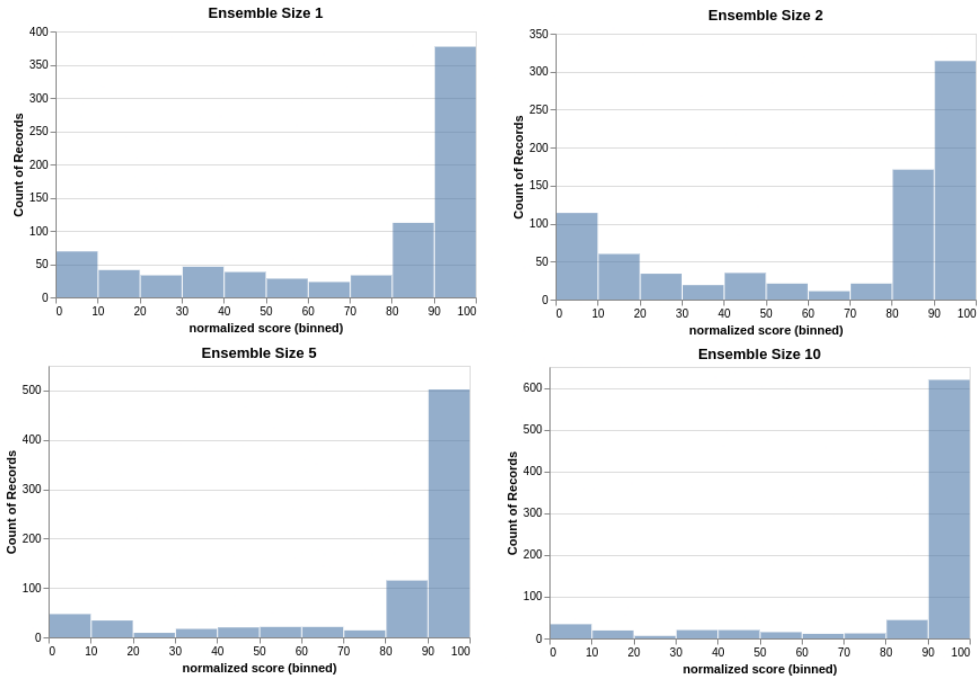


Figure 3.7: Performance of $\mathcal{I}_{\text{MG}}$ with varying ensemble sizes on Walker2d-Medium-Replay-v2. Each variant aggregates returns over 8 seeds, and each seed evaluates for 100 episodes. Each Q-value network is modeled by a 3-layer MLP. With lower ensemble size, the performance exhibits large variance across different episodes.

with an ensemble of Q, we obtain $\hat{Q}_0(s,a)$ by generalizing (3.63) as below

$$\begin{aligned} \hat{Q}_0(s,a) &= \hat{\mu}_Q^{\text{MLP}} - \sqrt{\frac{1}{M}\sum_{k=1}^{M}\left(\hat{Q}^{\text{MLP}}(s,a;\theta_k) - \hat{\mu}_Q^{\text{MLP}}\right)^2}, \\ \text{where} \quad \hat{\mu}_Q^{\text{MLP}} &= \frac{1}{M}\sum_{k=1}^{M}\hat{Q}^{\text{MLP}}(s,a;\theta_k). \end{aligned} \tag{3.64}$$

Other than the Q-value network, we applied the same setting as $\mathcal{I}_{\text{MG}}$ in Table 3.4. Fig. 3.6 presents the results with different ensemble sizes, showing that the performance generally increases with the ensemble size. Such a phenomenon illustrates a limitation of our CFPI operator $\mathcal{I}_{\text{MG}}$, as it heavily relies on accurate gradient information $\nabla_a[\hat{Q}_0(s,a)]_{a=a_\beta}$.

A large ensemble of Q is more likely to provide accurate gradient information, thus

leading to better performance. In contrast, a small ensemble size provides noisy gradient information, resulting in high variance across different rollout, as is shown in Fig. 3.7.

3.11.5 How to decide the number of gradient steps for SARSA training?

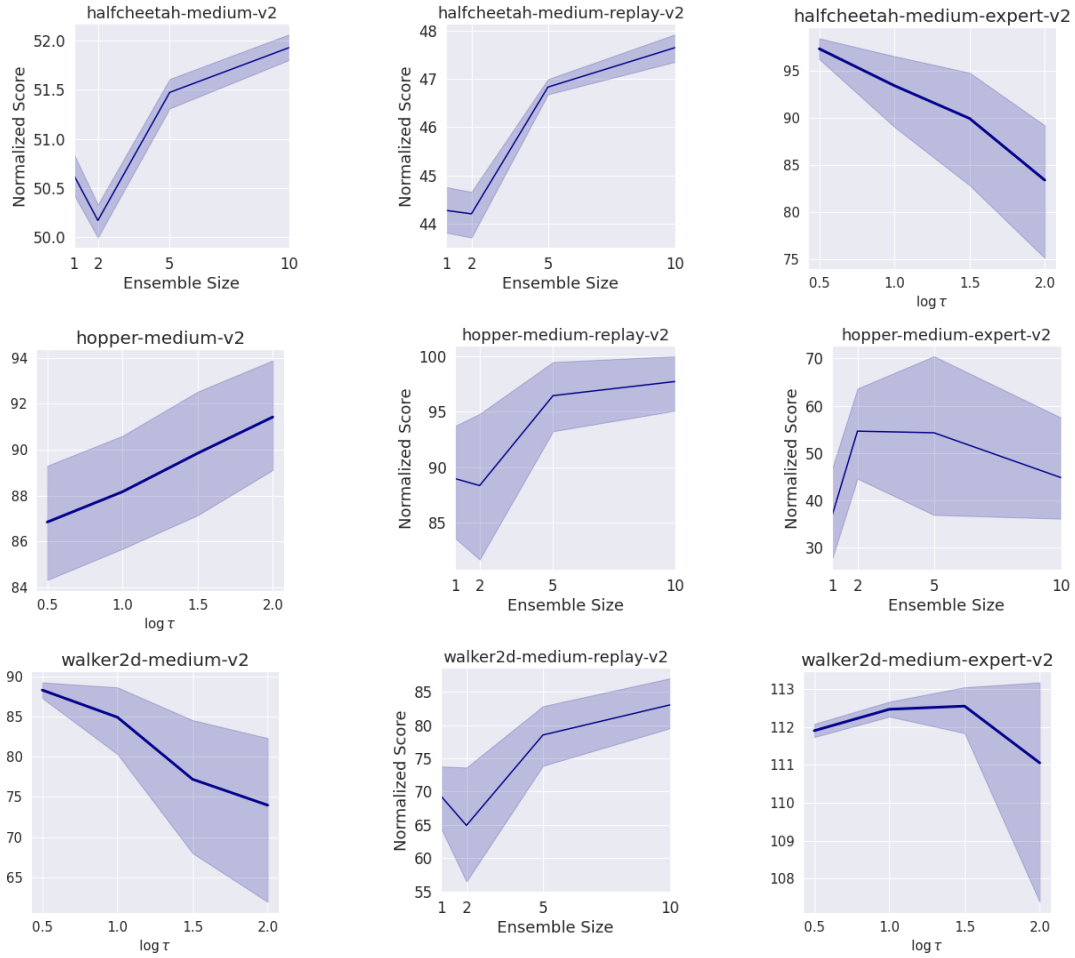


Figure 3.8: $\mathcal{L}_{\text{val}}$ on each dataset from the Gym-MuJoCo domain. We can observe that the model overfits to the training set when training for too may gradient steps. Each figure averages the validation loss over 2 folds with the same training seed. The shaded area denotes one standard deviation.

Deciding the number of gradient steps is a non-trivial problem in offline RL. While we use a fixed number of gradient steps for behavior cloning, we design a rigorous procedure to decide the gradient steps for SARSA training, inspired by the success of *k-fold validation*.

In our preliminary experiments, we first train a $\hat{Q}^{\beta}_{\text{all}}$ using all data from each dataset for 2M gradient steps. We model the $\hat{Q}^{\beta}_{\text{all}}(s, a)$ as a 3-layer MLP and train following

Table 3.16: Gradient steps for the SARSA training

Dataset	Gradient steps (K)
HalfCheetah-Medium-v2	200
Hopper-Medium-v2	400
Walker2d-Medium-v2	700
HalfCheetah-Medium-Replay-v2	1500
Hopper-Medium-Replay-v2	300
Walker2d-Medium-Replay-v2	1100
HalfCheetah-Medium-Expert-v2	400
Hopper-Medium-Expert-v2	400
Walker2d-Medium-Expert-v2	400

subsection 3.11.4. By training in this way, we treat $\hat{Q}^{\beta}_{\text{all}}(s,a)$ as the ground truth $Q^{\beta}(s,a)$ for all (s,a) sampled the dataset $\mathcal{D}$. Next, we randomly split the dataset with the ratio 95/5 to create the trainining set $\mathcal{D}_{\text{train}}$ validation set $\mathcal{D}_{\text{val}}$. We then train a new $\hat{Q}^{\beta}$ the SARSA training on $\mathcal{D}_{\text{train}}$. Therefore, we can define the validation loss as

$$\mathcal{L}_{\text{val}} = \mathbb{E}_{(s,a)\sim\mathcal{D}_{\text{val}}}||\hat{Q}^{\beta}_{\text{all}}(s,a) - \hat{Q}^{\beta}(s,a)||^2 \tag{3.65}$$

Fig. 3.8 presents the $\mathcal{L}_{\text{val}}$ on each dataset from the Gym-MuJoCo domain. We can clearly observe that $\hat{Q}^{\beta}$ generally overfits the $\mathcal{D}_{\text{train}}$ when training for too many gradient steps. We evaluate over two folds with one seed. Therefore, we can decide the gradient steps of each dataset for the SARSA training according to the results in Fig. 3.8 as listed in Table 3.16.

3.11.6 Our reproduced IQL training curves

We use the PyTorch [87] Implementation of IQL from RLkit [62] to obtain its policy π_{IQL} and value function Q_{IQL}. We do not use the official implementation[3] open-sourced by the authors because our CFPI operators are also based on PyTorch. Fig. 3.9 presents our reproduced training curves of IQL on the 6 datasets from the AntMaze domain.

We note that the IQL paper[4] does not report consistent results in their paper for the offline experiment performance on the AntMaze, as is shown in Table 3.17. We suspect that these results are obtained from different sets of random seeds. Therefore, we can conclude that our reproduced results match the results reported in the IQL paper. We believe our reproduction results of IQL are reasonable, even if we do not use the official implementation open-sourced by the authors.

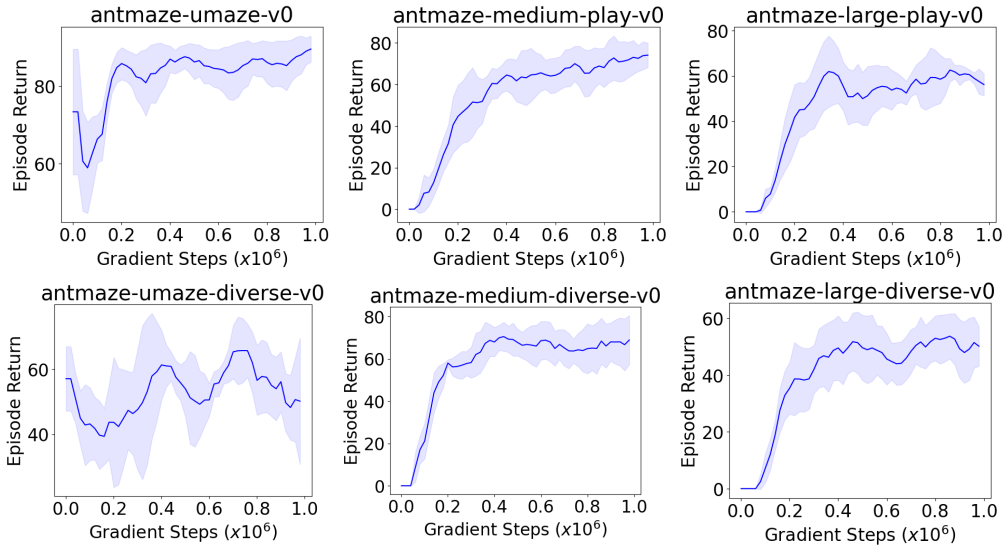


Figure 3.9: IQL offline training results on AntMaze. Shaded area denotes one standard deviation.

[3] `https://github.com/ikostrikov/implicit_q_learning`

[4] Link to the IQL paper. IQL's Table 5 & 6 are presented in the supplementary material.

Table 3.17: Offline experiment results on AntMaze reported in different tables from the IQL paper

Dataset	Table 1 & 6	Table 2	Table 5
antmaze-u-v0	87.5 ± 2.6	88.0	86.7
antmaze-u-d-v0	62.2 ± 13.8	67.0	75.0
antmaze-m-p-v0	71.2 ± 7.3	69.0	72.0
antmaze-m-d-v0	70.0 ± 10.9	71.8	68.3
antmaze-l-p-v0	39.6 ± 5.8	36.8	25.5
antmaze-l-d-v0	47.5 ± 9.5	42.2	42.6
Total	378.0 ± 49.9	374.8	370.1

3.11.7 Improve the policy learned by CQL

In this section, we show that our CFPI operators can also improve the policy learned by CQL [37] on the MuJoCo Gym Domain. We first obtain the CQL policy π_{CQL} and Q_{CQL} by training for 1M gradient steps using the official CQL implementation[5]. We obtain an improved policy $\mathcal{I}_{\text{SG}}(\pi_{\text{CQL}}, Q_{\text{CQL}}; \tau)$ that slightly outperforms π_{CQL} overall, as shown in Table 3.18. For all 6 tasks, we set $\log \tau = 0.1$.

Table 3.18: Improving the policy learned by IQL with our CFPI operator $\mathcal{I}_{\text{SG}}$

Dataset	π_{CQL} (1M)	$\mathcal{I}_{\text{SG}}(\pi_{\text{CQL}}, Q_{\text{CQL}})$
HalfCheetah-Medium-v2	45.5 ± 0.3	$\mathbf{47.1 \pm 1.5}$
Hopper-Medium-v2	65.4 ± 3.5	$\mathbf{70.1 \pm 4.9}$
Walker2d-Medium-v2	81.4 ± 0.6	$\mathbf{81.6 \pm 1.1}$
HalfCheetah-Medium-Replay-v2	44.6 ± 0.5	$\mathbf{45.9 \pm 1.7}$
Hopper-Medium-Replay-v2	95.2 ± 2.0	94.6 ± 1.6
Walker2d-Medium-Replay-v2	80.1 ± 2.6	78.8 ± 3.2
Total	412.2 ± 9.4	$\mathbf{418.2 \pm 13.9}$

[5]https://github.com/aviralkumar2907/CQL

3.12 Conclusion and Limitations

Motivated by the behavior constraint in the BCPO paradigm, we propose CFPI operators that perform policy improvement by solving an approximated BCPO in closed form. As practical datasets are usually generated by heterogeneous policies, we use the Gaussian Mixture to model the data-generating policies and overcome extra optimization difficulties by leveraging the LogSumExp's LB and Jensen's Inequality. We instantiate both one-step and iterative offline RL algorithms with our CFPI operator and show that they can outperform SOTA algorithms on the D4RL benchmark.

Our CFPI operators avoid the training instability incurred by policy improvement through SGD. However, our method still requires learning a good Q function. Specifically, our operators rely on the gradient information provided by the Q, and its accuracy largely impacts the effectiveness of our policy improvement. Therefore, one promising future direction for this chapter is to investigate ways to robustify the policy improvement given a noisy Q.

Chapter 4

Scaling Diffusion for Offline RL

4.1 Introduction

Generalist agents are characterized by the ability to plan over long horizons, understand and respond to human feedback, and generalize to new tasks based on that feedback. Language conditioning is an intuitive way to specify tasks, with a built-in structure enabling generalization to new tasks. There have been many recent developments to leverage language for robotics and downstream decision-making and control [88, 20, 89, 90, 91, 92]. However, while language is useful for task specification and generalization, it will not necessarily help with planning over long horizons.

Current methods have a few pitfalls. Many existing language-conditioned control methods assume access to a high-level discrete action space (e.g. switch on the stove, walk to the kitchen) provided by a lower level skill oracle [88, 94, 95, 20]. The generative large language model (LLM) will typically decompose some high-level language instruction into a set of predefined skills, which are then executed by a control policy or oracle. However, a fixed set of predefined skills may preclude the ability to generalize to novel environments and tasks. In addition, one of the most important design considerations for

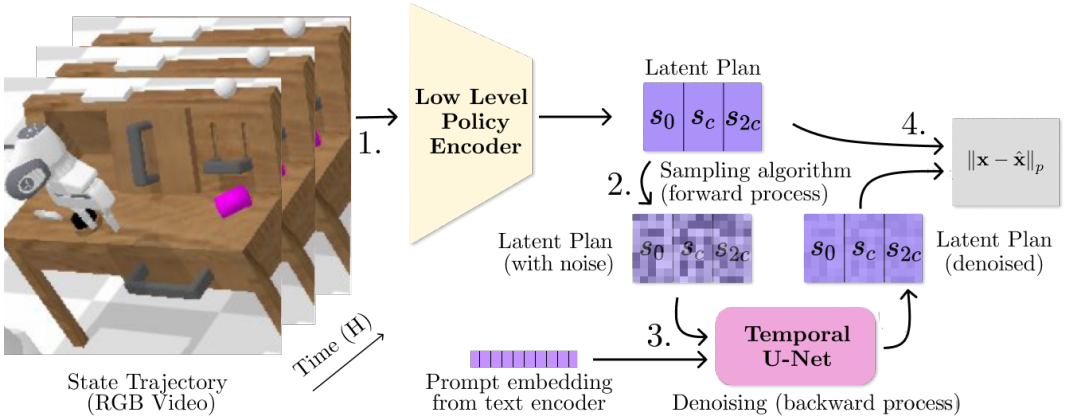


Figure 4.1: An overview of our high-level policy training pipeline. The frozen low-level policy encoder is used to encode a subsampled sequence of RGB observations into a lower dimensional latent space (1), which will be used later on as goals for the goal-conditioned low-level policy. We then noise this latent plan according to a uniformly sampled timestep from the diffusion process' variance schedule (2), and train a Temporal U-Net conditioned on natural language embeddings from a frozen upstream large language model to reverse the noising process (3), effectively learning how to conditionally denoise the latent plan. To train the U-Net, one can simply use the p-norm between the predicted latent plan and the ground truth latent plan as the loss (4). We use $p = 1$ in practice following [93].

these models is deciding what level of abstraction the communication protocol between the LLM and the policy should take. A tradeoff arises from this question, where one must decide between the amount of reasoning to distribute over the LM versus the underlying policy. We can increase the burden on the LLM by having it output text at a low-level of abstraction that is extremely simple to follow, or we can increase the burden on the policy by having it execute text at a higher level of abstraction. Much prior work has been done on increasing the reasoning load of the language encoder by leveraging the capabilities of LLMs to generate planning code [96], calculate affordances of separate skills [88], or perform chain of thought reasoning during rollout [97], but scaling the low-level policy for interpreting higher level instruction has remained relatively underexplored, an essential property necessary when the LLM fails to reason or gives overly broad instructions.

One promising candidate that has emerged for long horizon planning is denoising

diffusion models. Text-to-image diffusion models [98, 7, 99, 100] have recently been able to successfully take advantage of LLMs to generate incredibly detailed scenes. In addition, diffusion models have recently been proposed and successfully applied for low-dimensional, long-horizon planning and offline reinforcement learning (RL) [93, 101]. In particular, Diffuser [93] has emerged as an especially promising planner with several properties suited for language conditioned control: flexibility in task specification, temporal compositionality, and ability to scale to long-horizon settings.

However, Diffuser [93] does not work directly out of the box when scaling to pixels. This is perhaps unsurprising, as a high-dimensional Diffuser is effectively a text-to-video diffusion model, and even internet-scale video diffusion models have demonstrated only mediocre understanding of physics and temporal coherence over fine details [102, 103]. This is because the training objective for generative models has no notion of the underlying task, meaning they must model the entire scene with equal weighting. This includes potentially task-irrelevant details that may hinder or even prevent solving the actual control problem [104, 105], which leads to catastrophic failure in control where fine details and precision are essential. Additionally, training a video diffusion model is generally computationally expensive and may take several days or weeks to train, which leaves such an approach out of reach to most researchers [6]. This problem is exacerbated when considering that at inference time multiple forward passes (often >20) are required for generating even a single state and action, meaning full text to video diffusion models are inefficient and likely impractical for the real-time sampling demands of robotics and control. In summary, diffusion models are computationally prohibitive to run on high-dimensional input spaces and also tend to be inaccurate at low-level control.

We address both of these issues by proposing the usage of a hierarchical diffusion policy. By utilizing the representation of a goal-conditioned policy as a low-level policy (LLP), we can effectively solve both the representation and efficiency issue by outputting states

in a low-dimensional goal space directly into the LLP. This also allows us to arbitrarily scale the difficulty of the low-level policy learning problem by controlling the horizon length from which the goal state is set from the current state. This direction is promising, as it avoids defining the communication layer altogether and enables generating actions directly from high-level text without a human-defined communication protocol in between. In consequence, this hierarchical approach allows us to scale the diffusion model along three orthogonal axes: the **Spatial dimension** through a low-dimensional representation that has been purposely optimized for control, the **Time dimension** through a temporal abstraction enabled by utilizing a goal conditioned low-level policy (LLP), as the LLP can use goal states several timesteps away from the current state, and the **Task dimension** through language, as the diffusion model acts as a powerful interpreter for plugging any large language model into control. In addition, the entire pipeline is extremely fast and simple to train, as we utilize DDIM [106], a temporal abstraction on the horizon, as well as a low-dimensional representation for generation. We are able to achieve an average of 88.7% success rate across all tasks on the challenging CALVIN benchmark. Additionally, we elucidate where diffusion models for text-to-control work well and highlight their limitations. Finally, we explore the grounding between language and state-actions via evaluation of the task generalization capabilities of our hierarchical diffusion policy.

In summary, our core contributions are: **1)** We propose an effective method for improving diffusion policies' scaling to high-dimensional state spaces, longer time horizons, and more tasks by incorporating language and by scaling to pixel-based control. **2)** We significantly improve both the training and inference time of diffusion policies through DDIM, temporal abstraction, and careful analysis and choice of image encoder. **3)** A successful instantiation of our language control diffusion model that substantially outperforms the state of the art on the challenging CALVIN benchmark.

4.2 The Language Control Diffusion (LCD) Framework

In this section, we develop the Language Control Diffusion framework to enable scaling to longer horizons, improve the generalization capabilities of current language conditioned policies by avoiding the usage of a predefined low level skill oracle, and sidestep the computational prohibitiveness of training diffusion models for control from high-dimensional state spaces by proposing the usage of a vastly more efficient hierarchical RL framework. We start by deriving the high-level diffusion policy training objective, and go on to give a theoretical analysis on the suboptimality bounds of our approach by making the mild assumption of Lipschitz transition dynamics. We then solidify this theoretical framework into a practical algorithm by describing the implementation details of both the high-level and low-level policy. Here we also detail the key components of our method that most heavily affect training and inference efficiency, and the specific model architectures used in our implementation for the CALVIN benchmark [107].

4.2.1 Diffusion Policies in Hierarchical RL

High-level Diffusion Policy Objective. We first describe our problem formulation and framework in detail. Since we assume that our dataset is optimal, the policy objective reduces to imitation learning:

$$\min_{\pi} \mathbb{E}_{s,\mathcal{R}\sim\mathcal{D}} \left[D_{\mathrm{KL}} \left(\pi_\beta(\cdot \mid s, \mathcal{R}), \pi(\cdot \mid s, \mathcal{R}) \right) \right]. \tag{4.1}$$

As we tackle the problem from a planning perspective, we define a state trajectory generator as $\mathcal{P}$ and switch the atomic object from actions to state trajectories $\boldsymbol{\tau} = (s_0, s_1, ..., s_T)$.

Thus we aim to minimize the following KL:

$$\min_{\mathcal{P}} D_{\mathrm{KL}}\left(\mathcal{P}_\beta(\boldsymbol{\tau} \mid \mathcal{R}), \mathcal{P}(\boldsymbol{\tau} \mid \mathcal{R})\right) = \min_{\mathcal{P}} \mathbb{E}_{\boldsymbol{\tau}, \mathcal{R} \sim \mathcal{D}}\left[\log \mathcal{P}_\beta(\boldsymbol{\tau} \mid \mathcal{R}) - \log \mathcal{P}(\boldsymbol{\tau} \mid \mathcal{R})\right]. \tag{4.2}$$

This can be reformulated into the following diffusion training objective:

$$\min_{\theta} \mathbb{E}_{\boldsymbol{\tau}_0, \boldsymbol{\epsilon}}[\|\boldsymbol{\epsilon}_t - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\boldsymbol{\tau}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t, t)\|^2]. \tag{4.3}$$

Here t refers to a uniformly sampled diffusion process timestep, $\boldsymbol{\epsilon}_t$ refers to noise sampled from $\mathcal{N}(\mathbf{0}, \mathbf{I})$, $\boldsymbol{\tau}_0$ refers to the original denoised trajectory $\boldsymbol{\tau}$, α_t denotes a diffusion variance schedule and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$. We refer to section 4.6 for a more detailed derivation of our objective.

Near Optimality Guarantees. In order to theoretically justify our approach and show that we can safely do diffusion in the LLP encoder latent space without loss of optimality up to some error ϵ, we prove that a near optimal policy is always recoverable with the LLP policy encoder under some mild conditions: that our low-level policy has closely imitated our expert dataset and that the transition dynamics are Lipschitz smooth. Lipschitz transition dynamics is a fairly mild assumption to make [108, 109]. Several continuous control environments will exhibit a Lipschitz transition function, as well as many robotics domains. If an environment is not Lipschitz, it can be argued to be in some sense unsafe [110]. Moreover, one could then impose additional action constraints on such an environment to make it Lipschitz again.

In this chapter, we consider high dimensional control from pixels, and thus factorize our low level policy formulation into $\pi_{\mathrm{lo}}(a|s_t, g_t) := \phi(\mathcal{E}(s_t), g_t)$ where $z := \mathcal{E}(s_t)$ defines an encoder function that maps s into a latent representation z and ϕ translates the representation z into a distribution over actions a given a goal g. Note that ϕ is an

arbitrary function, and can be made as simple as a matrix or as complex as another hierarchical model. Let $\pi_{\text{lo}}(s) := \phi \circ \mathcal{E}(s)$ be a deterministic low-level policy. Similar to [111], we can then define the *sub-optimality* of the action and state space induced by $\tilde{\mathcal{A}} = \tilde{\mathcal{S}} = \mathcal{E}(s)$ to be the difference between the best possible high-level policy $\pi^*_{\text{hi}} = \arg\max_{\pi \in \Pi} J(\pi_{\text{hi}})$ within this new abstract latent space and the best policy in general:

$$\text{SubOpt}(\mathcal{E}, \phi) = \sup_{s \in S} V^{\pi^*}(s) - V^{\pi^*_{\text{hi}}}(s). \tag{4.4}$$

Intuitively, this captures how much potential performance we stand to lose by abstracting our state and action space.

Proposition 4.2.1. *If the transition function $p(s'|s,a)$ is Lipschitz continuous with constant K_f and $\sup_{s \in S, a \in A} |\pi_{\text{lo}}(s) - a^*| \leq \epsilon$, then*

$$\text{SubOpt}(\mathcal{E}, \phi) \leq \frac{2\gamma}{(1-\gamma)^2} R_{max} K_f \text{dom}(P(s'|s,a))\epsilon. \tag{4.5}$$

Crucially, this shows that the suboptimality of our policy is bounded by ϵ, and that our framework is able to recover a near-optimal policy with fairly mild assumptions on the environment. We give a detailed proof in section 4.7. The beauty of this result lies in the fact that due to the nature of the LLP's training objective of minimizing reconstruction error, we are directly optimizing for Proposition 4.2.1's second assumption. Moreover if this training objective converges, it achieves an appromixate π^*-irrelvance abstraction [112]. Therefore, utilizing the LLP encoder space for diffusion is guaranteed to be near-optimal as long as the LLP validation loss is successfully minimized.

Algorithm 4 Hierarchical Diffusion Policy Training

Input: baseline goal-conditioned policy $\pi_{\text{lo}} := \phi(\mathcal{E}(s_t), g_t)$, diffusion variance schedule α_t, temporal stride c, language model ρ
Output: trained hierarchical policy $\pi(a_t|s_t) := \pi_{\text{lo}}(a_t|s_t, \pi_{\text{hi}}(g_t|s_t))$, where g_t is sampled every c time steps from π_{hi} as the first state in $\boldsymbol{\tau}^c$.

Collect dataset $\mathcal{D}_{\text{onpolicy}}$ by rolling out trajectories $\boldsymbol{\tau} \sim \pi_{\text{lo}}, \rho$
Instantiate π_{hi} as diffusion model $\epsilon_\theta(\boldsymbol{\tau}_{\text{noisy}}, t, \rho(L))$
repeat
    Sample mini-batch $(\boldsymbol{\tau}, L) = B$ from $\mathcal{D}_{\text{onpolicy}}$.
    Subsample $\boldsymbol{\tau}^c = (\mathcal{E}(s_0), \mathcal{E}(s_c), \mathcal{E}(s_{2c}), ..., \mathcal{E}(s_T))$.
    Sample diffusion step $t \sim \text{Uniform}(\{1, ..., T\})$, noise $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
    Update high-level policy π_{hi} with gradient
    $-\nabla_\theta \|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\boldsymbol{\tau}^c + \sqrt{1-\bar{\alpha}_t}\epsilon, t, \rho(L))\|^2$
until converged

4.2.2 Practical Instantiation of Language Control Diffusion

We now describe our practical implementation and algorithm instantiation in this section. We first give an overview of our high-level policy instantiation before specifying the implementation of our low-level policy. Finally, we describe our model architecture in detail.

High-Level Policy. In Algorithm 4 we clarify the technical details of our high-level diffusion policy training. Before this point, we assume that a low-level policy (LLP) has been trained. Whilst our framework is flexible enough to use any LLP trained arbitrarily as long as it belongs to the factorized encoder family described in Section 4.2.1, in our practical implementation we first train our LLP through hindsight relabelling. After the policy has been trained, we then take the trained encoder representation and use it to compress the entire offline dataset of states. This is useful from a practical perspective as it caches the computation of encoding the state, often allowing for 100-200x less memory usage at train time. This enables significantly more gradient updates per second and

batch sizes, and is largely why our method is drastically more efficient than prior work utilizing diffusion for control. After caching the dataset with the pretrained encoder, we then induce a temporal abstraction by subsampling trajectories from the dataset, taking every c^{th} state in line 5 of Algorithm 4, where c refers to the temporal stride. This further increases efficiency and enables flexible restructuring of computational load between the high-level and low-level policy, simply by changing c. These states can then be trained with the typical diffusion training algorithm, detailed in lines 6-7.

Low-Level Policy. We adopt the HULC architecture [89] for our low-level policy. In HULC, the authors propose an improved version of the hierarchical Multi-Context Imitation Learning (MCIL). Note that this means that our method is a successful instantiation of a three-level hierarchical model, as we generate the goal states with diffusion *into their high-level policy*. HULC utilizes hierarchy by generating global discrete latent plans and learning local policies that are conditioned on this global plan. Their method focuses on several small components for increasing the effectiveness of text-to-control policies, including the architectures used to encode sequences in relabeled imitation learning, the alignment of language and visual representations, and data augmentation and optimization techniques.

Model Architecture. We adopt T5-XXL [113] as our textual encoder, which contains 11B parameters and outputs 4096 dimensional embeddings. T5-XXL has similarly found success in the text to image diffusion model Imagen [102]. We utilize a temporal U-Net [93], which performs 1D convolutions across the time dimension of the latent plan rather than the 2D convolution typical in text-to-image generation. This is motivated by our desire to preserve equivariance along the time dimension but not the state-action dimension. In addition, we modify the architecture in [93] by adding conditioning via cross attention in

a fashion that resembles the latent diffusion model [6]. Finally, we use DDIM [106] during inference for increased computational efficiency and faster planning. DDIM uses strided sampling and is able to capture nearly the same level of fidelity as typical DDPM sampling [99] with an order of magnitude speedup. For rolling out the latent plans generated by the denoiser, we resample a new sequence of goals g with temporal stride or frequency c, until either the task is completed successfully or the maximum horizon length is reached. Our low-level policy takes over control between samples, with goals generated by the high-level policy as input.

4.3 Experiments

In our experiments we aim to answer the following questions: 1) Does a diffusion-based approach perform well for language-conditioned RL? 2) How much efficiency is gained by planning in a latent space? Finally, 3) is a hierarchical instantiation of the diffusion model really necessary, and what tradeoffs are there to consider?

Table 4.1: **Our main result**. We compare success rates between our diffusion model and prior benchmarks on multitask long-horizon control (MT-LHC) for 34 disparate tasks. We report the mean and standard deviation across 3 seeds for our method with each seed evaluating for 1000 episodes. We bold the highest performing model in each benchmark category.

Horizon	GCBC	MCIL	HULC (LLP only)	Diffuser-1D	Diffuser-2D	Ours
One	64.7 ± 4.0	76.4 ± 1.5	82.6 ± 2.6	47.3 ± 2.5	37.4 ± 3.2	$\mathbf{88.7 \pm 1.5}$
Two	28.4 ± 6.2	48.8 ± 4.1	64.6 ± 2.7	18.8 ± 1.8	9.3 ± 1.3	$\mathbf{69.9 \pm 2.8}$
Three	12.2 ± 4.1	30.1 ± 4.5	47.9 ± 3.2	5.9 ± 0.4	1.3 ± 0.2	$\mathbf{54.5 \pm 5.0}$
Four	4.9 ± 2.0	18.1 ± 3.0	36.4 ± 2.4	2.0 ± 0.5	0.2 ± 0.0	$\mathbf{42.7 \pm 5.2}$
Five	1.3 ± 0.9	9.3 ± 3.5	26.5 ± 1.9	0.5 ± 0.0	0.07 ± 0.09	$\mathbf{32.2 \pm 5.2}$
Avg horizon len	1.11 ± 0.3	1.82 ± 0.2	2.57 ± 0.12	0.74 ± 0.03	0.48 ± 0.09	$\mathbf{2.88 \pm 0.19}$

4.3.1 Experimental Setup

Dataset and Metric. We evaluate on the CALVIN benchmark [107], a challenging multi-task, long-horizon robotics benchmark. After pretraining our low-level policy on all data, we freeze the policy encoder and train the diffusion temporal U-Net using the frozen encoder. We roll out all of our evaluated policies for 1000 trajectories on all 34 tasks, and all comparisons are evaluated in their official repository[1].

Baselines. As introduced in Section 4.2.2, we compare against the prior state of the art, HULC and MCIL [89, 114]. HULC provides a strong baseline as it is also a hierarchical method. MCIL (Multicontext Imitation Learning) uses a sequential CVAE to predict next actions from image or language-based goals, by modelling reusable latent sequences of states. GCBC (Goal-conditioned Behavior Cloning) simply performs behavior cloning without explictly modeling any latent variables like MCIL, and represents the performance of using the simplest low-level policy. Finally, Diffuser generates a trajectory through reversing the diffusion process. Diffuser is most comparable to our method, however, it utilizes no hierarchy. To ensure a fair comparison, we rigorously follow their evaluation procedure and build directly from their codebase. MCIL and GCBC results are taken from [89], whilst HULC results are reproduced from the original repository. Diffuser was retrained by following the original author's implementation.

4.3.2 Performance of LCD

We outperform prior methods on the challenging multi-task long-horizon control (MT-LHC) benchmark, and improve on the strongest prior model's average performance on horizon length one tasks by **6.1**% as shown in Table 4.1. In order to further elucidate why our method works better than HULC, we do a deeper analysis on several failure

[1] https://github.com/lukashermann/hulc/tree/fb14d5461ae54f919d52c0c30131b38f806ef8db

Table 4.2: Task generalization of LCD on a collection of five held out tasks. We test with 3 seeds and report the mean and std, evaluating on 20 rollouts per task for a total of 100 evaluations.

Task	Diffuser-1D	Ours
Lift Pink Block Table	31.67 ± 10.27	$\mathbf{55.00 \pm 16.33}$
Lift Red Block Slider	13.35 ± 10.25	$\mathbf{88.33 \pm 8.50}$
Push Red Block Left	1.64 ± 2.35	$\mathbf{35.00 \pm 7.07}$
Push Into Drawer	3.34 ± 4.71	$\mathbf{90.00 \pm 10.80}$
Rotate Blue Block Right	5.00 ± 4.10	$\mathbf{36.67 \pm 14.34}$
Avg SR	12.67 ± 3.56	$\mathbf{61.00 \pm 7.79}$

cases that we observed and show how LCD corrects them. In addition, in section 4.13 we give a breakdown of individual task success rates, where we find that LCD significantly outperforms HULC in 15 of the 34 tasks, outperforms HULC in 23 of the 34 tasks, and improves on the average success rate of single tasks by **3.33**%. Note that the average success rate differs from Table 4.1, as the distribution of tasks for MT-LHC is not uniform because the CALVIN benchmark filters out infeasible trajectories. A visualization of the first task MT-LHC distribution is provided for future work in section 4.9.

4.3.3 Task Generalization and Efficiency

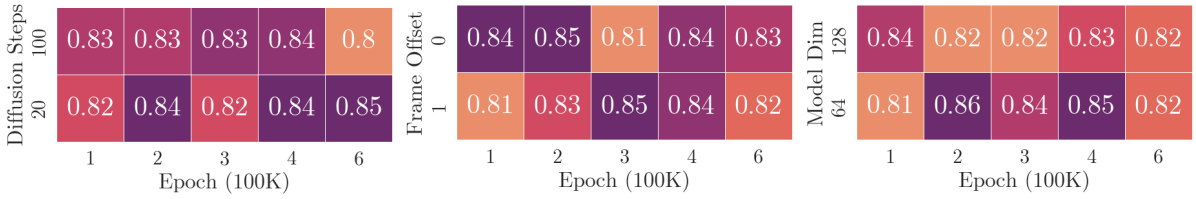
We test the task generalization of LCD on a collection of five held out tasks in Table 4.2. This benchmark is extremely difficult as it requires zero-shot generalization to a new task, which requires generalization across language, actions, and states. We find that LCD is able to successfully generalize, and is able to compose several disparate concepts from the training dataset together such as the color of the blocks, verbs such as lift and push, and object positions and state.

Through the usage of DDIM, temporal abstraction, and low-dimensional generation, we find in Table 4.3 that LCD is significantly faster during rollout and training than Diffuser. In addition, our method is significantly faster to train than HULC, albeit slower

Table 4.3: Wall clock times for training. Latent dims denotes the size of the latent space that we perform the diffusion generation in. We compare against two variants of Diffuser. Diffuser-1D is the same model as presented in Table 4.1 which utilizes a VAE trained from scratch on the dataset, whilst Diffuser-2D utilizes a large pretrained VAE from Stable Diffusion [6]. Inference time (sec) refers to the average amount of time taken in seconds to produce an action. LCD is 3.3x-15x faster during inference and 1.5x-3.7x faster during training compared to Diffuser-1D and Diffuser-2D.

	HULC	Diffuser-1D	Diffuser-2D	Ours (HLP only)	Ours (full)
Training (hrs)	82	20.8	49.2	**13.3**	95.3
Inference time (sec)	**0.005**	1.11	5.02	0.333	0.336
Model size	47.1M	74.7M	125.5M	**20.1M**	67.8M
Latent dims	N/A	256	1024	**32**	**32**
Avg ∇ updates/sec	.5	4	2.1	**6.25**	**6.25**

during rollout. This is to be expected, as HULC is not a diffusion model and only requires a single forward pass for generation. However, when comparing to other diffusion models we find that our method is 3.3x-15x faster during inference and 1.5x-3.7x faster during training. All numbers are taken from our own experiments and server for reproducibility and fairness, including baselines.



(a) Diffusion Steps (b) Frame Offset (c) Model Dim

Figure 4.2: Single task success rates from MT-LHC across different epochs and hyperparameters. Our diffusion model is robust to various hyperparameters such as diffusion steps, frame offset, hidden dimensions, and number of parameters.

4.3.4 Robustness to Hyperparameters

In Figure 4.2 we show that our diffusion method is robust to several hyperparameters including the number of diffusion steps/number of function evaluations (NFE), a frame offset o, and the hidden dimensions of the model. NFE is typically a key parameter

Table 4.4: Ablation of our method by comparing against a simple four layer MLP with 1024 hidden dimensions (4.8M total parameters) as high level policy. We use the same methodology as in Table 4.1, and report the mean and standard deviation across 3 seeds for our method with each seed evaluated for 1000 episodes.

Task	MLP	Ours
One	82.1 ± 4.0	$\mathbf{88.7 \pm 1.5}$
Two	61.0 ± 6.25	$\mathbf{69.9 \pm 2.8}$
Three	49.2 ± 4.15	$\mathbf{54.5 \pm 5.0}$
Four	32.1 ± 2.0	$\mathbf{42.7 \pm 5.2}$
Five	25.6 ± 0.9	$\mathbf{32.2 \pm 5.2}$
Avg SR	2.59 ± 0.01	$\mathbf{2.88 \pm 0.19}$

determining the quality of the representations. However, we found that the performance of our method is relatively insensitive to NFE. Frame offset o controls the spacing between goal state in the input data, which affects the temporal resolution of the representations. We consider augmenting our dataset by adding o during the sampling of our goal state s_{c+o} for potentially improving generalization. However, we find that this effect is more or less negligible. Finally, our method is also robust with respect to the number of parameters, and instantiating a larger model by increasing the model dimensions does not induce overfitting.

4.3.5 Is Diffusion Really Necessary?

In order to analyze whether diffusion actually improves HULC or if the gain comes just from the usage of a high-level policy, we perform an ablation study in Table 4.4 by using a simple MLP as a high-level policy, which receives the ground truth validation language embeddings as well as the current state, and attempts to predict the next goal state. An example of the ground truth language is "go push the red block right", which is never explicitly encountered during training. Instead, similar prompts of the same task are given, such as "slide right the red block". We find that even given the ground truth

validation language embeddings (the MLP does not need to generalize across language), this ablation significantly underperforms LCD, and slightly underperforms HULC. This suggests that our gains in performance over HULC truly do come from a better modeling of the goal space by diffusion.

4.3.6 Summary of Critical Findings

Diffuser fails to plan in original state space. After investigating the performance of diffuser, our findings in Figure 4.3 indicate that Diffuser fails to successfully replan in high dimensional state spaces when using a Variational Autoencoder (VAE). Based on our observations, we hypothesize that the failure of Diffuser to replan successfully is due to the diffusion objective enabling interpolation between low density regions in the VAE's latent space by the nature of the training objective smoothing the original probability distribution of trajectories. In practice, this interpolation between low density regions corresponds to physically infeasible trajectories. This suggests that interpolation between low density regions in the VAE's latent space is a significant factor in the failure of diffuser to plan and replan successfully. Our results highlight the need for better representation, and for further research of scaling diffusion models to high dimensional control.

Representation is essential for effective diffusion. We find that having a good representation is essential for effective diffusion in general control settings, as evidenced in Figure 4.3. A good representation greatly eases the learning difficulty on the diffusion model by reducing the number of dimensions needed to model, which in turn increases the information density and makes it easier for the model to learn the underlying dynamics of the system. The good representation likely massages the regions of low density latent space between feasible trajectories into areas that still correspond to physically feasible regions when decoded by the low level policy. This enables the model to generalize well

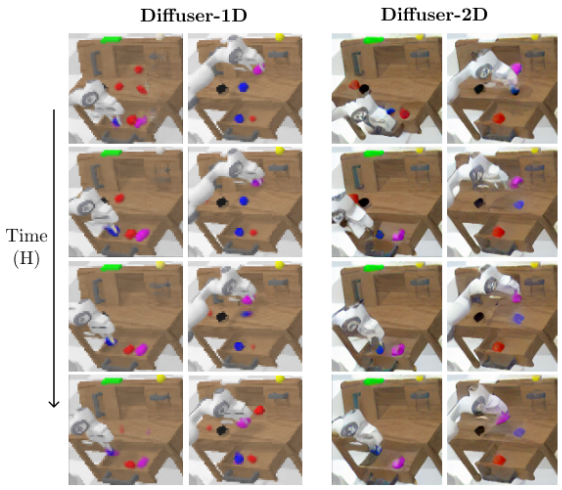


Figure 4.3: Denoised Latent Representations. Directly using latent diffusion models fails. Hallucination occurs on a β-TC VAE trained from scratch on the CALVIN dataset (Diffuser-1D), and loss of fine details occurs with SD v1.4's [6] internet-scale pretrained autoencoder (Diffuser-2D). For more and enlarged samples please refer to section 4.10.

to unseen situations, improving its robustness and flexibility. Additionally, by having a good representation, the diffusion model is able to successfully generalize to new scenarios, which is crucial for the model's effectiveness in a real-world application. Our results highlight the importance of latent representation in the diffusion process.

The encoder representation of deep goal-conditioned RL models can be effectively used for hierarchical RL. Our results imply that the encoder representation of deep reinforcement learning models have the potential to be effectively utilized in hierarchical reinforcement learning. In modern deep policy learning from pixels, we propose an effective task-aware representation can be extracted by using the latent space of an intermediate layer as an encoder. Although we have demonstrated the effectiveness of this approach by instantiating it successfully in a hierarchical diffusion model, this approach is general and can be applied to any generative model in hierarchical RL. Our results suggest that using a shared policy encoder between the high and low level policies

can improve the effectiveness and efficiency of generative modelling in hierarchical RL.

4.4 Related Work

Text-to-Control Models. Text-to-control models or language-conditioned policies [90, 91, 92] have been explored in the RL community for improving generalization to novel tasks and environments [115, 116, 117, 92, 118]. Although they are not the only way to incorporate external knowledge in the form of text to decision making tasks [119, 120], they remain one of the most popular [121, 122, 123]. However, language grounding in RL remains notoriously difficult, as language vastly underspecifies all possible configurations of a corresponding state [124]. Modern text to control models often still struggle with long-horizon language commands and misinterpret the language instruction. [125] attempt to solve a long-horizon Real-Time Strategy (RTS) game with a hierarchical method utilizing language as the communication interface between the high level and low level policy, whilst [23] consider training a language encoder-decoder for policy shaping, [126] utilize an attention module conditioned on textual entities for strong generalization and better language grounding. [127] propose a model-based objective to deal with sparse reward settings with language descriptions and [128] also tackle the sparse reward settings through the usage of intrinsic motivation with bonuses added on novel language descriptions. However, only [125] consider the long-horizon setting, and they do not consider a high-dimensional state space. [129] carefully examine the importance of diverse and massive data collection in enabling task generalization through language and propose a FiLM conditioned CNN-MLP model [130]. Much work has attempted the usage of more data and compute for control [131, 132, 133]. However, none of these methods consider the usage of diffusion models as the medium between language and RL.

Diffusion Models. Diffusion models such as DALL-E 2 [7] and GLIDE [98] have recently shown promise as generative models, with state-of-the-art text-to-image generation results demonstrating a surprisingly deep understanding of semantic relationships between objects and the high fidelity generation of novel scenes. Stable diffusion, an instantiation of latent diffusion [6], has also achieved great success and is somewhat related to our method as they also consider performing the denoising generation in a smaller latent space, albeit with a variational autoencoder [79] rather than a low level policy encoder. Given the success of denoising diffusion probabilistic models [99] in text-to-image synthesis [134], the diffusion model has been further explored in both discrete and continuous data domains, including image and video synthesis [135, 100], text generation [136], and time series [137]. Video generation models are especially relevant to this chapter, as they are a direct analogue of diffusion planning model Diffuser [93] in pixel space without actions. Diffuser first proposed to transform decision making into inpainting and utilize diffusion models to solve this problem, which much work has followed up on [138, 139, 140]. Specifically, they diffuse the state and actions jointly for imitation learning and goal-conditioned reinforcement learning through constraints specified through classifier guidance, and utilize Diffuser for solving long-horizon and task compositional problems in planning. Instead of predicting the whole trajectory for each state, [101] apply the diffusion model to sample a single action at a time conditioned by state. However, neither of these works considers control from pixels, or utilizing language for generalization.

4.5 Hyper-parameter settings and training details

For all methods we proposed in Table 4.1, Table 4.2, Table 4.6, and Table 4.5, we obtain the mean and standard deviation of each method across 3 seeds. Each seed contains the individual training process and evaluates the policy for 1000 episodes.

4.5.1 HP and training details for methods in Table 4.1 and Table 4.6.

Model	Module	Hyperparameter	Value
HULC	Trainer	Max Epochs	30
		β for KL Loss	0.01
		λ for Contrastive Loss	3
		Optimizer	Adam
		Learning Rate	2e-4
	Model	Transformer Hidden Size	2048
		Language Embedding Size	384
LCD	Gaussian Diffusion	Action Dimension	32
		Action Weight	10
		Loss Type	L2
		Observation Dimension	32
		Diffusion Steps	20
		Model Dimension	64
	Trainer	EMA Decay	0.995
		Label Frequency	200000
		Sample Frequency	1000
		Batch Size	512
		Learning Rate	2e-4
		Train Steps	250k
		Number of Steps Per Epoch	10000
		Normalizer	Gaussian Normalizer
		Frame Offset	0

Table 4.5: Hyperparameters for our methods in Table 4.1 and Table 4.6.

4.6 Training Objective Derivation

To model this, we turn to diffusion models [141], whom we borrow much of the following derivation from. Inspired by non-equilibrium thermodynamics, the common forms of diffusion models [142, 99, 106] propose modeling the data distribution $p(\boldsymbol{\tau})$ as a random process that steadily adds increasingly varied amounts of Gaussian noise to samples from $p(\boldsymbol{\tau})$ until the distribution converges to the standard normal. We denote the forward process as $f(\boldsymbol{\tau}_t|\boldsymbol{\tau}_{t-1})$, with a sequence of variances $(\beta_0, \beta_1 ... \beta_T)$. We define $\alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$.

$$f(\boldsymbol{\tau}_{1:T}|\boldsymbol{\tau}_0) = \prod_{t=1}^{T} f(\boldsymbol{\tau}_t|\boldsymbol{\tau}_{t-1}), \quad \text{where } f(\boldsymbol{\tau}_t|\boldsymbol{\tau}_{t-1}) = \mathcal{N}(\boldsymbol{\tau}_t; \sqrt{1-\beta_t}\boldsymbol{\tau}_{t-1}, \beta_t \mathbf{I}). \tag{4.6}$$

One can tractably reverse this process when conditioned on $\tau_{\mathbf{0}}$, which allows for the construction of a sum of the typical variational lower bounds for learning the backward process' density function [142]. Since the backwards density also follows a Gaussian, it suffices to predict μ_θ and Σ_θ which parameterize the backwards distribution:

$$p_\theta\left(\boldsymbol{\tau}_{t-1} \mid \boldsymbol{\tau}_t\right) = \mathcal{N}\left(\boldsymbol{\tau}_{t-1}; \boldsymbol{\mu}_\theta\left(\boldsymbol{\tau}_t, t\right), \boldsymbol{\Sigma}_\theta\left(\boldsymbol{\tau}_t, t\right)\right). \tag{4.7}$$

In practice, Σ_θ is often fixed to constants, but can also be learned through reparameterization. Following [99] we consider learning only μ_θ, which can be computed just as a function of τ_t and $\epsilon_\theta(\tau_t, t)$. One can derive that $\boldsymbol{\tau}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{\tau}_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ for $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, through a successive reparameterization of (4.6) until arriving at $f(\boldsymbol{\tau}_t|\boldsymbol{\tau}_0)$. Therefore to sample from $p(\boldsymbol{\tau})$, we need only to learn ϵ_θ, which is done by regressing to the ground truth ϵ given by the tractable backwards density. Assuming we have ϵ_θ, we can then follow a Markov chain of updates that eventually converges to the original data distribution, in a procedure reminiscent of Stochastic Gradient Langevin Dynamics [143]:

$$\boldsymbol{\tau}_{t-1} = \frac{1}{\sqrt{1-\beta_t}}\left(\boldsymbol{\tau}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta\left(\boldsymbol{\tau}_t, t\right)\right) + \sigma_t \mathbf{z}, \quad \text{where } \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}). \tag{4.8}$$

To learn ϵ_θ, we can minimize the following variational lower bound on the negative log-likelihood:

$$\begin{aligned}
L_{\text{CE}} &= -\mathbb{E}_{q(\boldsymbol{\tau}_0)} \log p_\theta(\boldsymbol{\tau}_0) \\
&= -\mathbb{E}_{q(\boldsymbol{\tau}_0)} \log \Big(\int p_\theta(\boldsymbol{\tau}_{0:T}) d\boldsymbol{\tau}_{1:T} \Big) \\
&= -\mathbb{E}_{q(\boldsymbol{\tau}_0)} \log \Big(\int q(\boldsymbol{\tau}_{1:T}|\boldsymbol{\tau}_0) \frac{p_\theta(\boldsymbol{\tau}_{0:T})}{q(\boldsymbol{\tau}_{1:T}|\boldsymbol{\tau}_0)} d\boldsymbol{\tau}_{1:T} \Big) \\
&= -\mathbb{E}_{q(\boldsymbol{\tau}_0)} \log \Big(\mathbb{E}_{q(\boldsymbol{\tau}_{1:T}|\boldsymbol{\tau}_0)} \frac{p_\theta(\boldsymbol{\tau}_{0:T})}{q(\boldsymbol{\tau}_{1:T}|\boldsymbol{\tau}_0)} \Big) \\
&\leq -\mathbb{E}_{q(\boldsymbol{\tau}_{0:T})} \log \frac{p_\theta(\boldsymbol{\tau}_{0:T})}{q(\boldsymbol{\tau}_{1:T}|\boldsymbol{\tau}_0)} \\
&= \mathbb{E}_{q(\boldsymbol{\tau}_{0:T})} \Big[\log \frac{q(\boldsymbol{\tau}_{1:T}|\boldsymbol{\tau}_0)}{p_\theta(\boldsymbol{\tau}_{0:T})} \Big] = L_{\text{VLB}}. \\
L_{\text{VLB}} &= L_T + L_{T-1} + \cdots + L_0 \\
\text{where } L_T &= D_{\text{KL}}(q(\boldsymbol{\tau}_T|\boldsymbol{\tau}_0) \parallel p_\theta(\boldsymbol{\tau}_T)) \\
L_t &= D_{\text{KL}}(q(\boldsymbol{\tau}_t|\boldsymbol{\tau}_{t+1}, \boldsymbol{\tau}_0) \parallel p_\theta(\boldsymbol{\tau}_t|\boldsymbol{\tau}_{t+1})) \\
&\qquad\qquad \text{for } 1 \leq t \leq T-1 \text{ and} \\
L_0 &= -\log p_\theta(\boldsymbol{\tau}_0|\boldsymbol{\tau}_1).
\end{aligned} \tag{4.9}$$

Which enables us to find a tractable parameterization for training, as the KL between two Gaussians is analytically computable.

$$
\begin{aligned}
L_t &= \mathbb{E}_{\boldsymbol{\tau}_0,\boldsymbol{\epsilon}}\Big[\frac{1}{2\|\boldsymbol{\Sigma}_\theta(\boldsymbol{\tau}_t,t)\|_2^2}\|\tilde{\boldsymbol{\mu}}_t(\boldsymbol{\tau}_t,\boldsymbol{\tau}_0)-\boldsymbol{\mu}_\theta(\boldsymbol{\tau}_t,t)\|^2\Big] \\
&= \mathbb{E}_{\boldsymbol{\tau}_0,\boldsymbol{\epsilon}}\Big[\frac{1}{2\|\boldsymbol{\Sigma}_\theta\|_2^2}\|\frac{1}{\sqrt{\alpha_t}}\Big(\boldsymbol{\tau}_t-\frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_t\Big)-\frac{1}{\sqrt{\alpha_t}}\Big(\boldsymbol{\tau}_t-\frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\boldsymbol{\tau}_t,t)\Big)\|^2\Big] \\
&= \mathbb{E}_{\boldsymbol{\tau}_0,\boldsymbol{\epsilon}}\Big[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\|\boldsymbol{\Sigma}_\theta\|_2^2}\|\boldsymbol{\epsilon}_t-\boldsymbol{\epsilon}_\theta(\boldsymbol{\tau}_t,t)\|^2\Big] \\
&= \mathbb{E}_{\boldsymbol{\tau}_0,\boldsymbol{\epsilon}}\Big[\frac{(1-\alpha_t)^2}{2\alpha_t(1-\bar{\alpha}_t)\|\boldsymbol{\Sigma}_\theta\|_2^2}\|\boldsymbol{\epsilon}_t-\boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\boldsymbol{\tau}_0+\sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t,t)\|^2\Big].
\end{aligned}
\tag{4.10}
$$

After removing the coefficient at the beginning of this objective following [99], we arrive at the objective used in the practical algorithm 1:

$$
\mathbb{E}_{\boldsymbol{\tau}_0,\boldsymbol{\epsilon}}[\|\boldsymbol{\epsilon}_t-\boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\boldsymbol{\tau}_0+\sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}_t,t)\|^2]. \tag{4.11}
$$

Furthermore, thanks to the connection between noise conditioned score networks and diffusion models [144, 99], we are able to state that $\epsilon_\theta \propto -\nabla \log p(\boldsymbol{\tau})$:

$$
\begin{aligned}
\mathbf{s}_\theta(\boldsymbol{\tau}_t,t) &\approx \nabla_{\boldsymbol{\tau}_t}\log p(\boldsymbol{\tau}_t) \\
&= \mathbb{E}_{q(\boldsymbol{\tau}_0)}[\nabla_{\boldsymbol{\tau}_t}p(\boldsymbol{\tau}_t|\boldsymbol{\tau}_0)] \\
&= \mathbb{E}_{q(\boldsymbol{\tau}_0)}\Big[-\frac{\boldsymbol{\epsilon}_\theta(\boldsymbol{\tau}_t,t)}{\sqrt{1-\bar{\alpha}_t}}\Big] \\
&= -\frac{\boldsymbol{\epsilon}_\theta(\boldsymbol{\tau}_t,t)}{\sqrt{1-\bar{\alpha}_t}}.
\end{aligned}
\tag{4.12}
$$

Therefore, by using a variant of ϵ_θ conditioned on language to denoise our latent plans, we can effectively model $-\nabla_\tau \mathcal{P}_\beta(\boldsymbol{\tau} \mid \mathcal{R})$ with our diffusion model, iteratively guiding our generated trajectory towards the optimal trajectories conditioned on language.

4.7 Proof of 4.2.1

Proof. The proof is fairly straightforward, and can be shown by translating our definition of suboptimality into the framework utilized by [111]. We are then able to leverage their first theorem bounding suboptimality by the Total Variation (TV) between transition distributions to show our result, as TV is bounded by the Lipschitz constant multiplied by the domain of the function.

[111] first define a low level policy generator Ψ which maps from $S \times \tilde{A}$ to Π. Using the high level policy to sample a goal $g_t \sim \pi_{\text{hi}}(g|s_t)$, they use Ψ to translate this to a policy $\pi_t = \Psi(s_t, g_t)$, which samples actions $a_{t+k} \sim \pi_t(a|s_{t+k}, k)$ from $k \in [0, c-1]$. The process is repeated from s_{t+c}. Furthermore, they define an inverse goal generator $\varphi(s, a)$, which infers the goal g that would cause Ψ to yield an action $\tilde{a} = \Psi(s, g)$. The following can then be shown:

Theorem 4.7.1. *If there exists* $\varphi : S \times A \to \tilde{A}$ *such that,*

$$\sup_{s \in S, a \in A} D_{TV}(P(s'|s,a)||P(s'|s, \Psi(s, \varphi(s,a)))) \leq \epsilon, \tag{4.13}$$

then $\text{SubOpt}'(\Psi) \leq C\epsilon$*, where* $C = \frac{2\gamma}{(1-\gamma)^2} R_{max}$.

Note that their SubOpt' is different from ours; whilst we defined in terms of the encoder $\mathcal{E}$ and action generator ϕ, they define it in terms of Ψ. Note, however, that the two are equivalent when the temporal stride $c = 1$, as Ψ becomes $\pi_{\text{lo}} = \phi \circ \mathcal{E}$. It is essential to note that when using a goal conditioned imitation learning objective, as we do in this chapter, π_{lo} becomes equivalent to an inverse dynamics model $\text{IDM}(s, \mathcal{E}(s)) = a$ and that $\varphi(s, a)$ becomes equivalent to $\mathcal{E}(s')$. This is the key to our proof, as the second

term in the total variation of 4.7.1 reduces to

$$\begin{aligned} & P(s'|s, \Psi(s, \varphi(s, a)))) \\ & = P(s'|s, \Psi(s, \mathcal{E}(s'))) \\ & = P(s'|s, a + \epsilon)). \end{aligned} \tag{4.14}$$

Since we have that the transition dynamics are Lipschitz:

$$\begin{aligned} & \int_{\mathcal{A},\mathcal{S}} |P(s'|s, a) - P(s'|s, a + \epsilon))| \; d\nu \\ & \leq \int_{\mathcal{A},\mathcal{S}} K_f |a - (a + \epsilon)| \, d\nu \\ & = K_f \epsilon \int_{\mathcal{A},\mathcal{S}} d\nu \\ & = K_f \epsilon \; \mathrm{dom}(P(s'|s, a)) \end{aligned} \tag{4.15}$$

Which we can then plug into 4.13 to obtain the desired $C = \frac{2\gamma}{(1-\gamma)^2} R_{max} K_f \mathrm{dom}(P(s'|s, a))$.

□

4.8 Diffuser-2D

Here we give details for our strongest Diffusion-based ablation, which uses Stable Diffusion's VAE for generating latent plans, which outputs a latent 2D feature map, with height and width 1/8 of the original image. Plans are sampled with a temporal stride of 7, such that each trajectory covers a total of 63 timesteps with $t = 0, 7, 14...63$. Overall, generation quality tends to be higher and more temporally coherent than that of the 1D model, but low level details still not precise enough for control from pixels. For examples of model outputs, please refer to subsection 4.10.2.

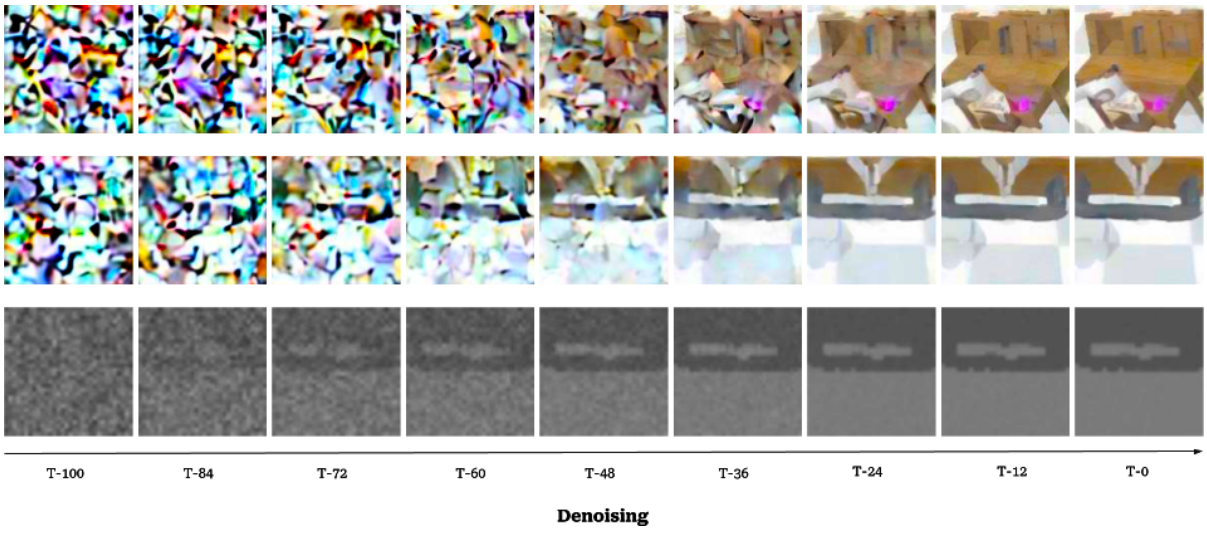


Figure 4.4: An overview of our Denoising process. In Figure 4.4, we give an example of the denoising process of one of our ablations, the Diffuser-2D model. This model utilizes the 2D autoencoder of [6] with [93].

4.9 Task Distribution

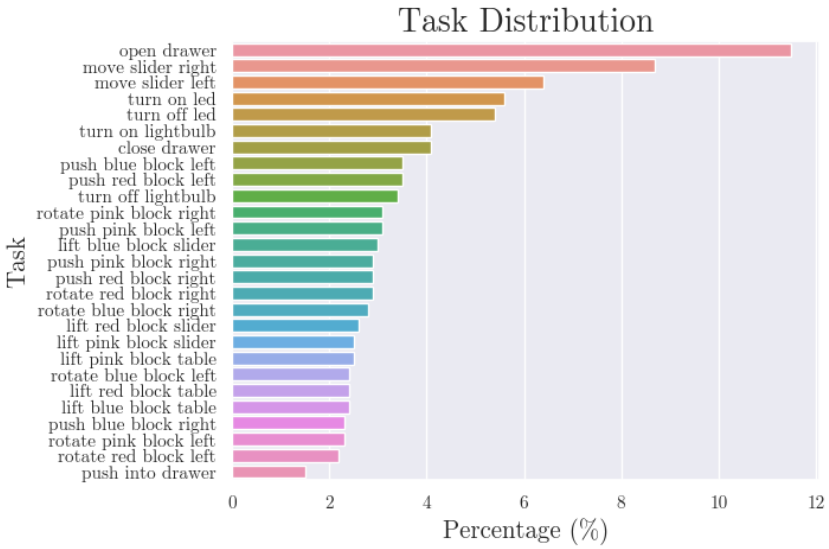


Figure 4.5: The Evaluation Task Distribution. We visualize the distribution of all the tasks considered in our experiments in Figure 4.5. Note the long-tailedness of this distribution, and how it skews evaluation scores upwards if one can solve the relatively easier tasks that occur most frequently, such as Open Drawer, Move Slider Right, and Move Slider Left. These tasks only deal with static objects, meaning there is very little generalization that is needed in order to solve these tasks when compared to other block tasks involving randomized block positions.

4.10 Representation Failures

4.10.1 Diffuser-1D (β-TC VAE Latent Representation) Failures

We give a few failure cases of decoded latent plans, where the latent space is given by a trained from scratch β-TC VAE on the CALVIN D-D dataset. The top row of each plan comes from the static camera view, whilst the bottom one comes from the gripper camera view (a camera located at the tool center point of the robot arm). The VAE is trained by concatenating the images in the channel dimension, and compressing to 128 latent dimensions. Plans are sampled with a temporal stride of 9, such that each trajectory covers a total of 63 timesteps with $t = 0, 9, 18...63$. Interestingly, we found that replanning during rollout did not work, precluding the possibility of success on CALVIN with our implementation of this method.

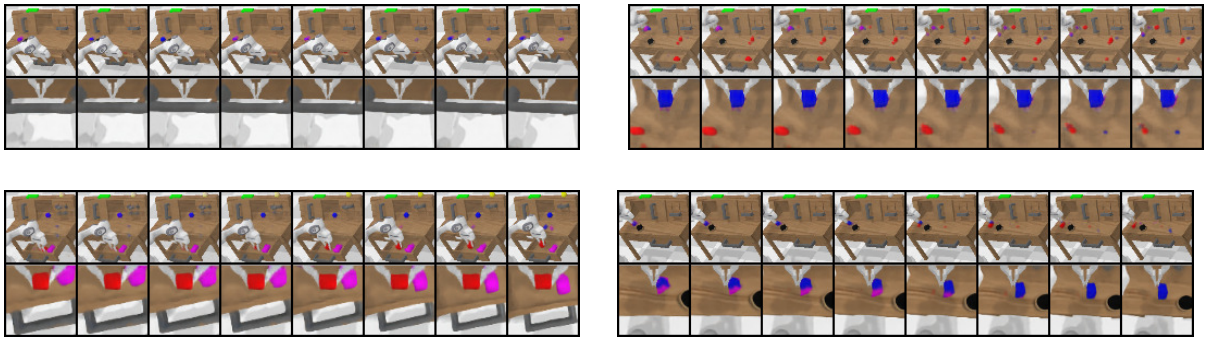


Figure 4.6: **(a)** An example of the Close Drawer Task. Notice the flickering block on the top right of the table. Also note the entangled red and blue blocks at the top left of the table. **(b)** An example of the Lift Blue Block Slider Task. The gripper view is temporally incoherent, red and blue blocks in slider are entangled. **(c)** An example of the Lift Red Block Drawer task. Two blocks begin to appear on the table at the end of generation. The red block is also not clearly generated in the first frame. **(d)** An example of the Push Blue Block Right task. The blue block on the table becomes red by the end of the static view, whereas the opposite happens in the gripper view.

4.10.2 Diffuser-2D Failures

We additionally give some failure cases for Diffuser-2D (Stable Diffusion Latent Representation). For more information on the training of this model, please refer to section 4.8. We also found that replanning during rollout did not work with this model.

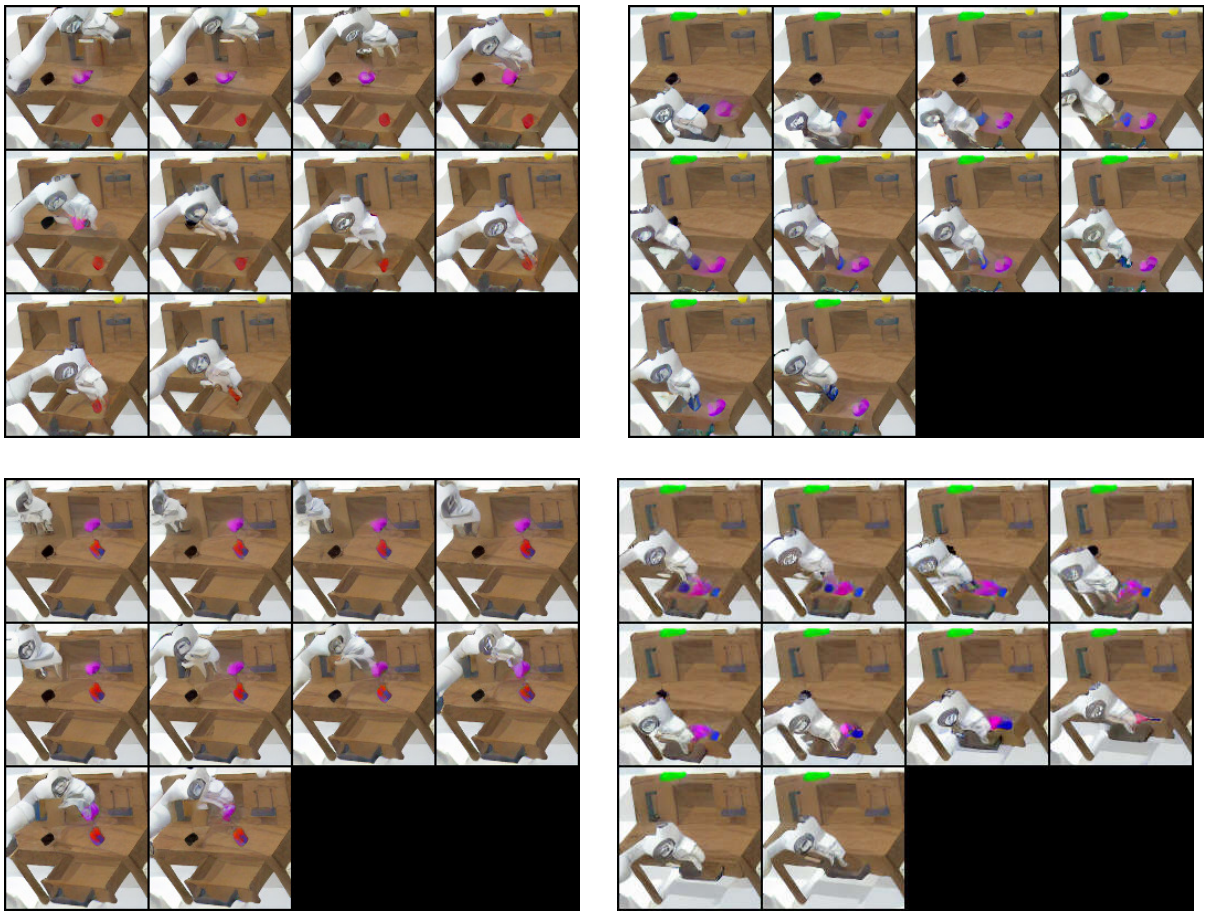


Figure 4.7: **(a)** An example of the Lift Red Block Drawer Task. Note the pink block that disappears. **(b)** An example of the Lift Blue Block Drawer Task. The gripper arm is entangled with the block. **(c)** An example of the Lift Pink Block Slider Task. Note the entangled red/blue blocks. **(d)** An example of the Close Drawer Task. Note the entangled pink/blue blocks.

4.11 TSNE Comparison between Groud Truth (GT) trajectory and Diffuser-1D (DM) trajectory

In order to better understand whether the representation failures found in section 4.10 are a result of the underlying encoder or the diffusion model, we visualize the TSNE embeddings of an encoded successful trajectory from the dataset, which we refer to as a Ground Truth trajectory, and the TSNE embeddings of generated trajectories from Diffuser-1D (DM) in Figure 4.8. If we observe that the DM's embedddings are fairly close to the GT-VAE's, then we can reasonably presume that the VAE is the failure mode, whereas if the trajectories are wildly different this would imply that the DM is failing to model the VAE's latent distribution properly. Here, all samples other than 6 appear to fairly close, so we suspect that the failure case lies in the underlying latent distribution and not the DM's modeling capabilities. This is further backed by LCD, as we show that by using the proper underlying latent space with a LLP leads to success.

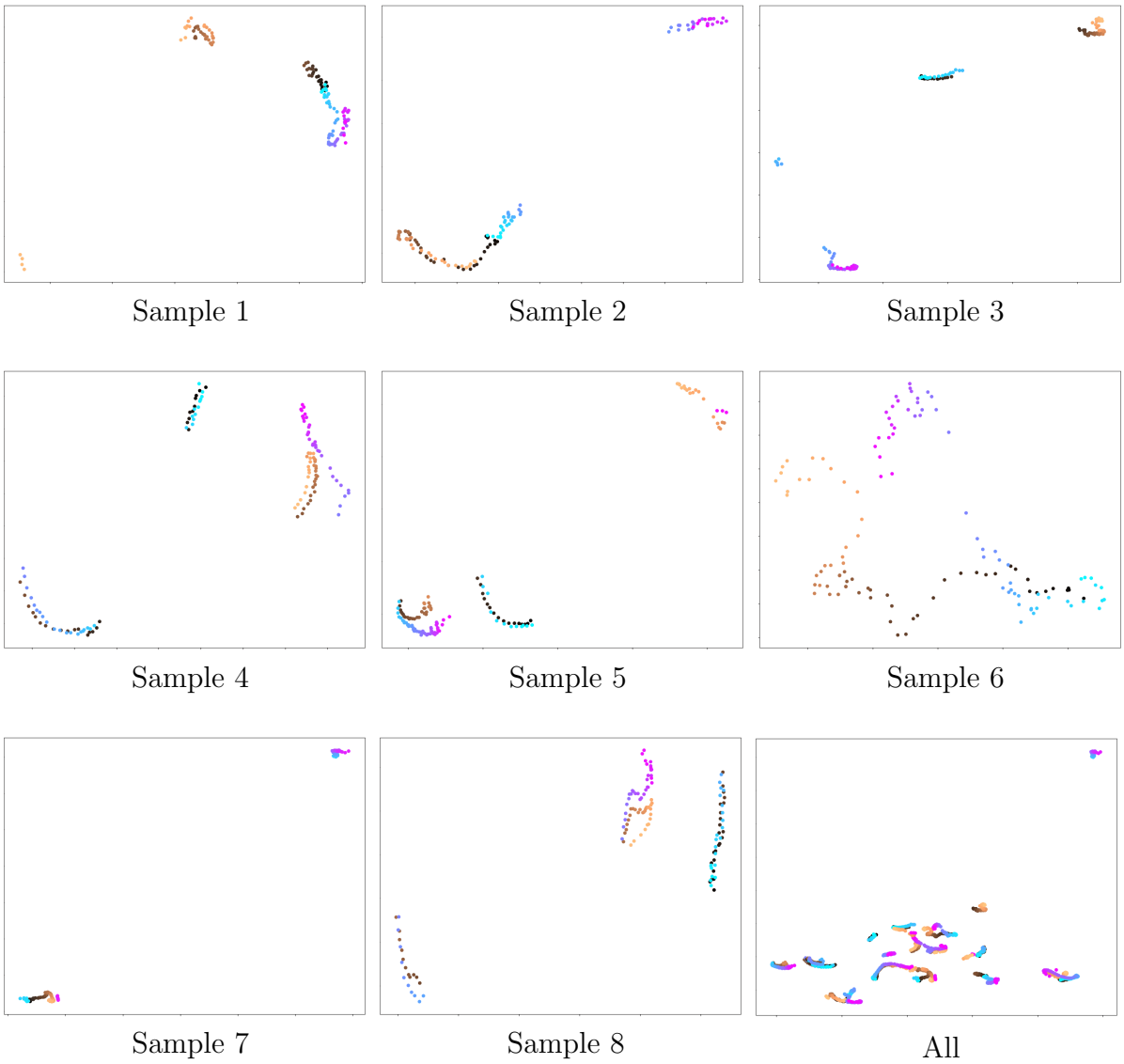


Figure 4.8: TSNE visualization of GT-VAE trajectory vs. Diffuser-1D trajectory, where the purple and light blue color range is the ground truth VAE, and the copper color range is Diffuser-1D. All states are normalized, and all trajectories are taken from the task "lift pink block table".

4.12 HULC Latent Plan TSNE

We give TSNE embeddings of several Latent Plans generated during inference by HULC below.

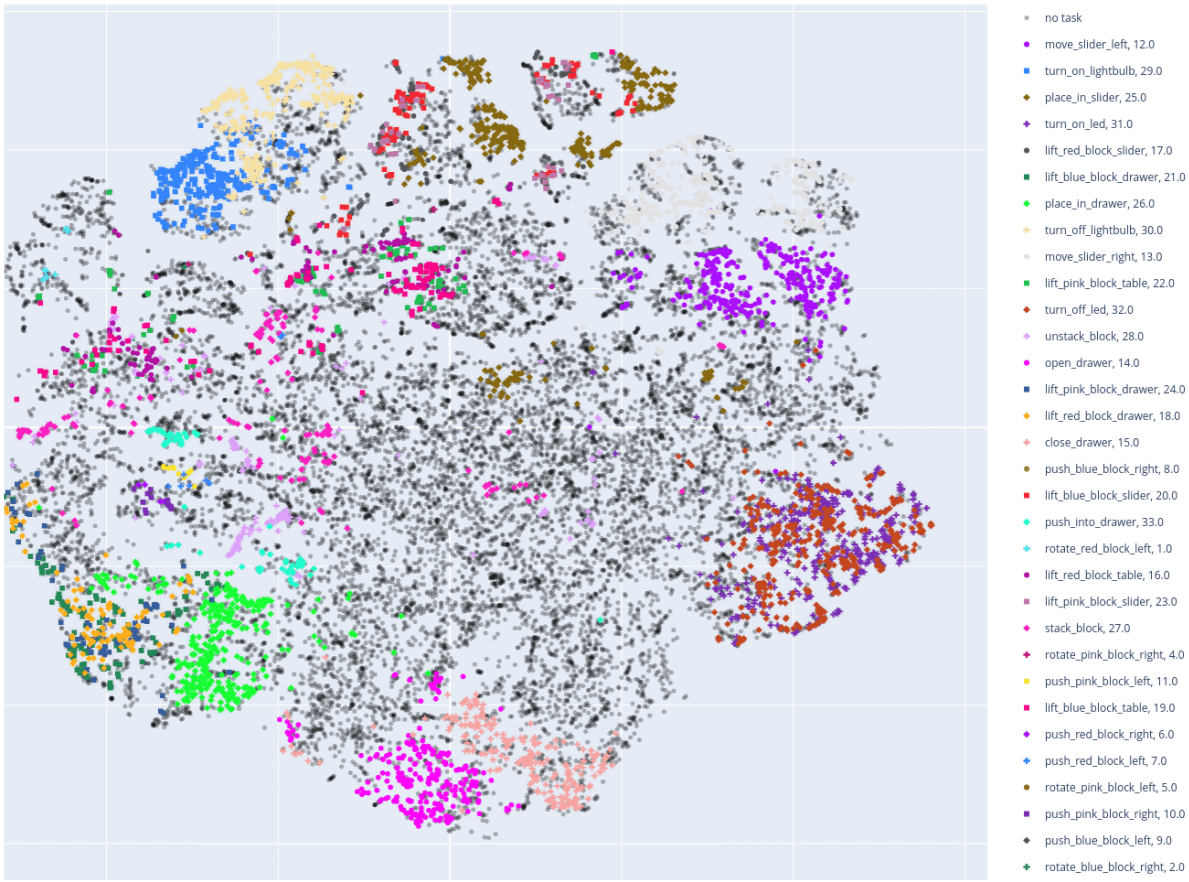


Figure 4.9: TSNE of Latent Plan. We give a TSNE embedding of the latent plan space of HULC in Figure 4.9. The latent plan space is the communication layer between the high level policy and low level policy of the HULC model, which corresponds to the intermediate layer between the lower level and lowest level policy in our method. We clarify that this is not the latent goal space that our model does generation in. Our method performs latent generation in the earlier layer from the output of the goal encoder, which corresponds to 32 latent dimensions.

4.13 Comparison of success rates (SR) across single tasks, evaluated with HULC and LCD

Table 4.6: Comparison of success rates (SR) across single tasks, evaluated with HULC and LCD. We again report the performance for the mean and standard deviation across 3 seeds for all methods, with runs taken from the MT-LHC benchmark in Table 4.1.

Task	HULC	Ours
Close Drawer	99.50 ± 0.36	95.97 ± 3.46
Lift Blue Block Drawer	86.97 ± 13.01	90.83 ± 9.47
Lift Blue Block Slider	58.80 ± 6.99	$\mathbf{78.77 \pm 5.20}$
Lift Blue Block Table	72.73 ± 7.89	76.07 ± 2.03
Lift Pink Block Drawer	67.93 ± 9.39	76.27 ± 22.80
Lift Pink Block Slider	71.50 ± 4.54	$\mathbf{79.30 \pm 2.01}$
Lift Pink Block Table	68.00 ± 5.60	70.47 ± 3.07
Lift Red Block Drawer	$\mathbf{91.67 \pm 11.79}$	77.00 ± 4.71
Lift Red Block Slider	71.97 ± 2.57	79.70 ± 7.32
Lift Red Block Table	60.87 ± 5.84	$\mathbf{69.43 \pm 2.77}$
Move Slider Left	96.27 ± 1.14	96.27 ± 1.96
Move Slider Right	99.27 ± 0.24	97.67 ± 2.19
Open Drawer	97.83 ± 1.43	98.07 ± 0.50
Place In Drawer	95.93 ± 1.37	95.90 ± 1.79
Place In Slider	70.90 ± 5.44	$\mathbf{81.10 \pm 0.14}$
Push Blue Block Left	60.10 ± 9.69	71.93 ± 6.82
Push Blue Block Right	27.60 ± 9.10	33.10 ± 2.86
Push Into Drawer	72.13 ± 5.74	75.47 ± 5.29
Push Pink Block Left	66.73 ± 17.27	64.80 ± 11.98
Push Pink Block Right	57.67 ± 4.46	51.17 ± 6.04
Push Red Block Left	64.17 ± 4.71	55.30 ± 16.32
Push Red Block Right	28.73 ± 16.07	37.83 ± 3.27
Rotate Blue Block Left	59.63 ± 7.38	$\mathbf{70.03 \pm 2.56}$
Rotate Blue Block Right	62.57 ± 4.17	$\mathbf{72.77 \pm 6.76}$
Rotate Pink Block Left	71.53 ± 6.99	75.07 ± 9.15
Rotate Pink Block Right	55.93 ± 4.29	$\mathbf{64.00 \pm 4.84}$
Rotate Red Block Left	67.43 ± 5.15	$\mathbf{81.23 \pm 8.86}$
Rotate Red Block Right	69.53 ± 5.44	74.57 ± 2.11
Stack Block	$\mathbf{36.83 \pm 2.84}$	32.00 ± 3.56
Turn Off Led	98.30 ± 1.87	96.87 ± 1.80
Turn Off Lightbulb	98.33 ± 1.25	95.67 ± 3.78
Turn On Led	98.73 ± 0.93	97.63 ± 1.73
Turn On Lightbulb	$\mathbf{98.97 \pm 0.40}$	95.60 ± 1.58
Unstack Block	94.53 ± 4.53	90.80 ± 3.72
Unstack Block	94.53 ± 4.53	90.80 ± 3.72
Avg Success Rate	70.68 ± 2.65	74.01 ± 2.64

4.14 Model Card for Language Control Diffusion

A hierarchical diffusion model for long horizon language conditioned planning.

4.14.1 Model Details

Model Description

A hierarchical diffusion model for long horizon language conditioned planning.

4.14.2 Uses

Direct Use

Creating real world robots, controlling agents in video games, solving extended reasoning problems from camera input

Downstream Use

Could be deconstructed so as to extract the high level policy for usage, or built upon further by instantiating a multi-level hierarchical policy

Out-of-Scope Use

Discrimination in real-world decision making, military usage

4.14.3 Bias, Risks, and Limitations

Significant research has explored bias and fairness issues with language models (see, e.g., Sheng et al. (2021) and Bender et al. (2021)). Predictions generated by the model may include disturbing and harmful stereotypes across protected classes; identity characteristics; and sensitive, social, and occupational groups.

4.14.4 Training Details

Training Data

http://calvin.cs.uni-freiburg.de/

4.14.5 Environmental Impact

Carbon emissions can be estimated using the Machine Learning Impact calculator presented in Lacoste et al. (2019). Baselines are run with either 8 Titan RTX or 8 A10 GPUs following the original author guidelines, whilst our experiments are run with a single RTX or A10. In total this project used around 9000 hours of compute.

- **Hardware Type:** NVIDIA Titan RTX, NVIDIA A10
- **Hours used:** 9000
- **Cloud Provider:** AWS
- **Compute Region:** us-west-2
- **Carbon Emitted:** 1088.64 kg

4.14.6 Technical Specifications

Model Architecture and Objective

Temporal U-Net, Diffusion objective

Compute Infrastructure

Hardware Nvidia Titan RTX , Nvidia A10

Software Pytorch

4.15 Conclusion and Limitations

Learning atomic sub-skills through language is critical to scaling to more complex and open environments. We explore solving this problem through learned state and temporal abstractions, and show that the strengths of diffusion models can be leveraged for long horizon plans, and their weaknesses at low-level detail generation can be managed through learning low-level, goal-conditioned policies with imitation learning. Experiments and qualitative analysis demonstrate the simplicity and effectiveness of our model, showing that LCD can achieve state-of-the-art performance on a competitive language-conditioned control benchmark from rich observations, over long horizons, and generalize to new scenarios. For future work, one could extend LCD to incorporate additional levels of hierarchy and scale to even longer horizons. Additionally, a deeper analysis on the interplay of different representations and the diffusion model could be performed. Finally, one could further probe the task generalization, and potentially improve generalization on the language side by leveraging better pre-trained models and incorporating that distilled knowledge to improve reasoning and planning for control.

Chapter 5

Conclusion

5.1 Contribution

In conclusion, this thesis has introduced and explored the concept of "Closed-Form Policy Improvement" and "Language Control Diffusion", innovative methods that employ the insights gleaned from Natural Language Processing (NLP). We've seen how harnessing the lessons of scaling data, models, and transformer architectures from NLP can yield significant empirical results in the realm of Reinforcement Learning, particularly Offline RL.

5.2 Limitations

However, a critical inquiry must be made: **Are we treading the right path?** The pursuit of AGI is fraught with several profound challenges and potential pitfalls that must be carefully examined. Firstly, the approach of naive scaling may not necessarily culminate in the creation of AGI. There may exist fundamental limitations - possibly ones that we can't even conceive of yet - which could impede our efforts, making AGI an

unreachable dream.

Secondly, we face the issue of the 'black box' phenomenon. Our current approach, while showing promising results, lacks transparency and explainability. As these models grow increasingly complex, it becomes more challenging to understand how they make decisions. This opacity can seriously impact their robustness, trustworthiness, and ultimately, their acceptance and integration into society.

The environmental cost of our quest for AGI presents a third critical concern. For instance, the carbon footprint of training a model like GPT-3 is equivalent to around 91 years of an average human's life [145, 146] – an alarming statistic that calls for urgent redressal. As we scale models, the environmental implications scale proportionately, posing a serious threat to our planet.

Lastly, the societal impact of these models, while potentially transformative, is not necessarily positive. They hold the potential to enable mass misinformation, leading to socio-political unrest. In a military context, these models could be misused to create weapons of mass destruction, threatening global peace. Economically, the benefits of AGI might only further widen the existing wealth gap, as those with access to this technology could amass disproportionate economic advantages.

Hence, as we tread the path to AGI, we must stay vigilant and considerate of these potential hurdles. The pursuit of AGI is as much an ethical and societal endeavor as it is a technological one. Therefore, the principles that guide our journey must be underpinned by a deep commitment to sustainability, transparency, fairness, and above all, the betterment of humanity. As we stride ahead in this quest, let's ensure that we're not just creating intelligent machines, but we're also creating a better world.

A

Appendix

A.1 Table of Abbreviations

Full Name	Abbreviation
Artificial General Intelligence	AGI
Reinforcement Learning	RL
Markov Decision Processes	MDP
Natural Language Processing	NLP
Quadratically Constrained Linear Program	QCLP
Behavior Constrained Policy Optimization	BCPO
Out-of-distribution	OOD
Stochastic Gradient Descent	SGD
Closed-Form Policy Improvement	CFPI
State-of-the-art	SOTA
Single Gaussian	SG
Gaussian Mixture	MG
Single Gaussian CFPI Operator	$\mathcal{I}_{\mathrm{SG}}(\pi_\beta, Q; \tau)$
Gaussian Mixture CFPI Operator	$\mathcal{I}_{\mathrm{MG}}(\pi_\beta, Q; \tau)$
Function Divergence Measure	$D(\cdot, \cdot)$
Kullback-Leibler Divergence	$D_{\mathrm{KL}}(\cdot, \cdot)$
Action	a
Behavior Action	a_β
Policy	$\pi(\cdot\|s)$
Behavior Policy	$\pi_\beta(\cdot\|s)$
Optimal Policy	$\pi^\star(\cdot\|s)$
Distance Threshold	τ

Full Name	Abbreviation
Language Control Diffusion	LCD
Composing Actions from Language and Vision	CALVIN
Low-Level Policy	LLP
Denoising Diffusion Implicit Model	DDIM
Hierarchical Universal Language Conditioned Policy	HULC
Multi-context Imitation Learning	MCIL
Goal-conditioned Behavior Cloning	GCBC
Multitask Long Horizon Control	MT-LHC
Multilayered Perceptron	MLP
Variational Autoencoder	VAE
β-Total Correlation	β-TC
Hierarchical RL	HRL
Error	ϵ
Inverse Dynamics Model	IDM
Ground Truth	GT
Diffusion Model	DM
Success Rate	SR
t-distributed stochastic neighbor embedding	TSNE
Error	ϵ
State Space	$\mathcal{S}$
Action Space	$\mathcal{A}$
Reward Function	$\mathcal{R}$
Transition Function	T
Discount Factor	γ
Initial state distribution	ρ_0
Timestep	t
Dataset	$\mathcal{D}$
Trajectory	$\boldsymbol{\tau}$
Error	ϵ
Diffusion Variance Schedule	$\beta_i := 1 - \alpha_i$
Normal Distribution	$\mathcal{N}(x; \mu, \Sigma)$.
Mean	μ
Covariance Matrix	Σ
State Trajectory Generator	$\mathcal{P}$
Encoder	$\mathcal{E}$
Latent State Representation	z
Action Decoder	$\phi(z, g_t)$
Temporal Stride	c
Language Model	ρ
Language-to-reward Operator	$\psi : \mathcal{L} \mapsto \mathcal{F}$

A.2 List of Figures

3.1 Aggregate metrics [63] with 95% CIs based on results reported in Table 3.4. The CIs are estimated using the percentile bootstrap with stratified sampling. Higher median, IQM, and mean scores, and lower Optimality Gap correspond to better performance. Our $\mathcal{I}_{\text{MG}}$ outperforms baselines by a significant margin based on all four metrics. section 3.9 includes additional details. 30

3.2 Iterative $\mathcal{I}_{\text{MG}}$ training results on AntMaze. Shaded area denotes one standard deviation. 54

3.3 Comparison between our methods and baselines using reliable evaluation methods proposed in [63]. We re-examine the results in Table 3.4 on the 9 tasks from the D4RL MuJoCo Gym domain. Each metric is calculated with a 95% CI bootstrap based on 9 tasks and 10 seeds for each task. Each seed further evaluates each method for 100 episodes. The interquartile mean (IQM) discards the top and bottom 25% data points and calculates the mean across the remaining 50% runs. The IQM is more robust as an estimator to outliers than the mean while maintaining less variance than the median. Higher median, IQM, mean scores, and lower Optimality Gap correspond to better performance. Our $\mathcal{I}_{\text{MG}}$ outperforms the baseline methods by a significant margin based on all four metrics. 57

3.4 Performance profiles (score distributions) for all methods on the 9 tasks from the D4RL MuJoCo Gym domain. The average score is calculated by averaging all runs within one task. Each task contains 10 seeds, and each seed evaluates for 100 episodes. Shaded area denotes 95% confidence bands based on percentile bootstrap and stratified sampling [63]. The η value where the curves intersect with the dashed horizontal line $y = 0.5$ corresponds to the median, while the area under the performance curves corresponds to the mean. 58

3.5 Performance of $\mathcal{I}_{\text{MG}}$ with varying $\log \tau$. The other HP can be found in Table 3.8. Each variant averages returns over 10 seeds, and each seed contains 100 evaluation episodes. The shaded area denotes bootstrapped 95% CI. 68

3.6 Performance of $\mathcal{I}_{\text{MG}}$ with varying ensemble sizes. Each variant averages returns over 8 seeds, and each seed contains 100 evaluation episode. Each Q-value network is modeled by a 3-layer MLP. The shaded area denotes bootstrapped 95% CI. 70

3.7 Performance of $\mathcal{I}_{\text{MG}}$ with varying ensemble sizes on Walker2d-Medium-Replay-v2. Each variant aggregates returns over 8 seeds, and each seed evaluates for 100 episodes. Each Q-value network is modeled by a 3-layer MLP. With lower ensemble size, the performance exhibits large variance across different episodes. 71

3.8 $\mathcal{L}_{\text{val}}$ on each dataset from the Gym-MuJoCo domain. We can observe that the model overfits to the training set when training for too may gradient steps. Each figure averages the validation loss over 2 folds with the same training seed. The shaded area denotes one standard deviation. 73

3.9 IQL offline training results on AntMaze. Shaded area denotes one standard deviation. 75

4.1 An overview of our high-level policy training pipeline. The frozen low-level policy encoder is used to encode a subsampled sequence of RGB observations into a lower dimensional latent space (1), which will be used later on as goals for the goal-conditioned low-level policy. We then noise this latent plan according to a uniformly sampled timestep from the diffusion process' variance schedule (2), and train a Temporal U-Net conditioned on natural language embeddings from a frozen upstream large language model to reverse the noising process (3), effectively learning how to conditionally denoise the latent plan. To train the U-Net, one can simply use the p-norm between the predicted latent plan and the ground truth latent plan as the loss (4). We use $p = 1$ in practice following [93]. 79

4.2 Single task success rates from MT-LHC across different epochs and hyperparameters. Our diffusion model is robust to various hyperparameters such as diffusion steps, frame offset, hidden dimensions, and number of parameters. 90

4.3 Denoised Latent Representations. Directly using latent diffusion models fails. Hallucination occurs on a β-TC VAE trained from scratch on the CALVIN dataset (Diffuser-1D), and loss of fine details occurs with SD v1.4's [6] internet-scale pretrained autoencoder (Diffuser-2D). For more and enlarged samples please refer to section 4.10. 93

4.4 An overview of our Denoising process. In Figure 4.4, we give an example of the denoising process of one of our ablations, the Diffuser-2D model. This model utilizes the 2D autoencoder of [6] with [93]. 102

4.5 The Evaluation Task Distribution. We visualize the distribution of all the tasks considered in our experiments in Figure 4.5. Note the long-tailedness of this distribution, and how it skews evaluation scores upwards if one can solve the relatively easier tasks that occur most frequently, such as Open Drawer, Move Slider Right, and Move Slider Left. These tasks only deal with static objects, meaning there is very little generalization that is needed in order to solve these tasks when compared to other block tasks involving randomized block positions. 103

4.6 **(a)** An example of the Close Drawer Task. Notice the flickering block on the top right of the table. Also note the entangled red and blue blocks at the top left of the table. **(b)** An example of the Lift Blue Block Slider Task. The gripper view is temporally incoherent, red and blue blocks in slider are entangled. **(c)** An example of the Lift Red Block Drawer task. Two blocks begin to appear on the table at the end of generation. The red block is also not clearly generated in the first frame. **(d)** An example of the Push Blue Block Right task. The blue block on the table becomes red by the end of the static view, whereas the opposite happens in the gripper view. 104

4.7 **(a)** An example of the Lift Red Block Drawer Task. Note the pink block that disappears. **(b)** An example of the Lift Blue Block Drawer Task. The gripper arm is entangled with the block. **(c)** An example of the Lift Pink Block Slider Task. Note the entangled red/blue blocks. **(d)** An example of the Close Drawer Task. Note the entangled pink/blue blocks. 105

4.8 TSNE visualization of GT-VAE trajectory vs. Diffuser-1D trajectory, where the purple and light blue color range is the ground truth VAE, and the copper color range is Diffuser-1D. All states are normalized, and all trajectories are taken from the task "lift pink block table". 107

4.9 TSNE of Latent Plan. We give a TSNE embedding of the latent plan space of HULC in Figure 4.9. The latent plan space is the communication layer between the high level policy and low level policy of the HULC model, which corresponds to the intermediate layer between the lower level and lowest level policy in our method. We clarify that this is not the latent goal space that our model does generation in. Our method performs latent generation in the earlier layer from the output of the goal encoder, which corresponds to 32 latent dimensions. 108

A.3 List of Tables

3.1 Comparison between our one-step policy and SOTA methods on the Gym-MuJoCo domain of D4RL. Our method uses the same τ for all datasets except Hopper-M-E (detailed in subsection 3.10.1). We report the mean and standard deviation of our method's performance across 10 seeds. Each seed contains an individual training process and evaluates the policy for 100 episodes. We use Cheetah for HalfCheetah, M for Medium, E for Expert, and R for Replay. We bold the best results for each task. 27

3.2 Comparison between our Iterative $\mathcal{I}_{\text{MG}}$ and SOTA methods on the AntMaze domain. We report the mean and standard deviation across 5 seeds for our method with each seed evaluating for 100 episodes. The performance for all baselines is directly reported from the IQL paper. Our Iterative $\mathcal{I}_{\text{MG}}$ outperforms all baselines on 5 out of 6 tasks and obtains the best overall performance. 28

3.3 Our $\mathcal{I}_{\text{SG}}(\pi_{\text{IQL}}, Q_{\text{IQL}})$ improves the IQL policy π_{IQL} on AntMaze. We report the mean and standard deviation of 10 seeds. Each seed evaluates for 100 episodes. 29

3.4 Ablation studies of our Method on the Gym-MuJoCo domain. Again we report the mean and standard deviation of 10 seeds, and each seed evaluates for 100 episodes. Our $\mathcal{I}_{\text{MG}}$ outperforms baselines by a significant margin. At the same time, the SGD-based method Rev. KL Reg exhibits substantial performance variations, demonstrating the importance of a stable policy improvement operator. 30

3.5 Comparison between our iterative algorithm and SOTA methods on the AntMaze domain of D4RL. We report the mean and standard deviation across 5 seeds for our methods. Our Iterative $\mathcal{I}_{\text{MG}}$ outperforms all baselines on 5 out of 6 tasks and obtaining the best overall performance, demonstrating the effectiveness of our CFPI operator when instantiating an iterative algorithm. 53
3.6 Hyperparameters for our Iterative $\mathcal{I}_{\text{MG}}$. 55
3.7 $\mathcal{I}_{\text{DET}}$ results on the Gym-MuJoCo domain. We report the mean and standard deviation 5 seeds and each seed evaluates for 100 episodes. . . . 56
3.8 Hyperparameters for our methods in Table 3.1 and Table 3.4. 59
3.9 HP search for MG-EBCQ. We report the mean and std of 10 seeds, and each seed evaluates for 100 episodes. 63
3.10 HP search for SG-EBCQ. We report the mean and std of 10 seeds, and each seed evaluates for 100 episodes. 64
3.11 HP search for MG-Rev. KL Reg. We report the mean and std of 10 seeds, and each seed evaluates for 100 episodes. 64
3.12 HP search for SG-Rev. KL Reg. We report the mean and std of 10 seeds, and each seed evaluates for 100 episodes. 65
3.13 Hyperparameters for methods in Table 3.3 66
3.14 Results of MG-MS on the MuJoCo Gym domain. We report the mean and standard deviation across 10 seeds, and each seed evaluates for 100 episodes. 67
3.15 Comparison between setting the number of Gaussian components to 4 and 8 for our $\mathcal{I}_{\text{MG}}$ on the three Medium-Replay datasets. We report the mean and standard deviation across 10 seeds, and each seed evaluates for 100 episodes. 69
3.16 Gradient steps for the SARSA training 74

3.17 Offline experiment results on AntMaze reported in different tables from the IQL paper 76
3.18 Improving the policy learned by IQL with our CFPI operator $\mathcal{I}_{\text{SG}}$ 76

4.1 **Our main result**. We compare success rates between our diffusion model and prior benchmarks on multitask long-horizon control (MT-LHC) for 34 disparate tasks. We report the mean and standard deviation across 3 seeds for our method with each seed evaluating for 1000 episodes. We bold the highest performing model in each benchmark category. 87
4.2 Task generalization of LCD on a collection of five held out tasks. We test with 3 seeds and report the mean and std, evaluating on 20 rollouts per task for a total of 100 evaluations. 89
4.3 Wall clock times for training. Latent dims denotes the size of the latent space that we perform the diffusion generation in. We compare against two variants of Diffuser. Diffuser-1D is the same model as presented in Table 4.1 which utilizes a VAE trained from scratch on the dataset, whilst Diffuser-2D utilizes a large pretrained VAE from Stable Diffusion [6]. Inference time (sec) refers to the average amount of time taken in seconds to produce an action. LCD is 3.3x-15x faster during inference and 1.5x-3.7x faster during training compared to Diffuser-1D and Diffuser-2D. 90
4.4 Ablation of our method by comparing against a simple four layer MLP with 1024 hidden dimensions (4.8M total parameters) as high level policy. We use the same methodology as in Table 4.1, and report the mean and standard deviation across 3 seeds for our method with each seed evaluated for 1000 episodes. 91
4.5 Hyperparameters for our methods in Table 4.1 and Table 4.6. 96

4.6 Comparison of success rates (SR) across single tasks, evaluated with HULC and LCD. We again report the performance for the mean and standard deviation across 3 seeds for all methods, with runs taken from the MT-LHC benchmark in Table 4.1. 109

A.4 List of Algorithms

1 Offline RL with CFPI operators 21

2 Iterative $\mathcal{I}_{\text{MG}}$ 52

3 Policy improvement of $\mathcal{I}_{\text{DET}}$ with a stochastic π_β 54

4 Hierarchical Diffusion Policy Training 85

Bibliography

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, *Scaling laws for neural language models*, *arXiv preprint arXiv:2001.08361* (2020).

[3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, *Attention is all you need*, in *Advances in Neural Information Processing Systems* (I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds.), vol. 30, Curran Associates, Inc., 2017.

[4] S. Fujimoto, D. Meger, and D. Precup, *Off-policy deep reinforcement learning without exploration*, in *International Conference on Machine Learning*, pp. 2052–2062, 2019.

[5] J. Li, E. Zhang, M. Yin, Q. Bai, Y.-X. Wang, and W. Y. Wang, *Offline reinforcement learning with closed-form policy improvement operators*, 2023.

[6] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, *High-resolution image synthesis with latent diffusion models*, in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022.

[7] A. Ramesh, P. Dhariwal, A. Nichol, C. Chu, and M. Chen, *Hierarchical text-conditional image generation with clip latents*, 2022.

[8] E. Zhang, Y. Lu, W. Wang, and A. Zhang, *Language control diffusion: Efficiently scaling through space, time, and tasks*, *arXiv* (2023).

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

[10] S. Fujimoto and S. S. Gu, *A minimalist approach to offline reinforcement learning*, in *Thirty-Fifth Conference on Neural Information Processing Systems*, 2021.

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

[12] D. Brandfonbrener, W. F. Whitney, R. Ranganath, and J. Bruna, *Offline rl without off-policy evaluation*, *arXiv preprint arXiv:2106.08909* (2021).

[13] N. Jaques, A. Ghandeharioun, J. H. Shen, C. Ferguson, A. Lapedriza, N. Jones, S. Gu, and R. Picard, *Way off-policy batch deep reinforcement learning of implicit human preferences in dialog*, *arXiv preprint arXiv:1907.00456* (2019).

[14] A. M. Metelli, M. Papini, F. Faccio, and M. Restelli, *Policy optimization via importance sampling*, *Advances in Neural Information Processing Systems* **31** (2018).

[15] A. M. Metelli, M. Papini, N. Montali, and M. Restelli, *Importance sampling techniques for policy optimization*, *The Journal of Machine Learning Research* **21** (2020), no. 1 5552–5626.

[16] A. Hallak, D. Di Castro, and S. Mannor, *Contextual markov decision processes*, *arXiv preprint arXiv:1502.02259* (2015).

[17] D. Abel, W. Dabney, A. Harutyunyan, M. K. Ho, M. Littman, D. Precup, and S. Singh, *On the Expressivity of Markov Reward*, in *Advances in Neural Information Processing Systems* (M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. W. Vaughan, eds.), vol. 34, pp. 7799–7812, Curran Associates, Inc., 2021.

[18] P. De Haan, D. Jayaraman, and S. Levine, *Causal confusion in imitation learning*, *Advances in Neural Information Processing Systems* **32** (2019).

[19] S. Ross, G. Gordon, and D. 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*, pp. 627–635, JMLR Workshop and Conference Proceedings, 2011.

[20] S. Li, X. Puig, Y. Du, C. Wang, E. Akyurek, A. Torralba, J. Andreas, and I. Mordatch, *Pre-trained language models for interactive decision-making*, *arXiv preprint arXiv:2202.01771* (2022).

[21] S. Arora and P. Doshi, *A survey of inverse reinforcement learning: Challenges, methods and progress*, 2018.

[22] S. Sodhani, A. Zhang, and J. Pineau, *Multi-task reinforcement learning with context-based representations*, 2021.

[23] B. Harrison, U. Ehsan, and M. O. Riedl, *Guiding reinforcement learning exploration using natural language*, *arXiv preprint arXiv:1707.08616* (2017).

[24] T. Schaul, D. Horgan, K. Gregor, and D. Silver, *Universal value function approximators*, in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 1312–1320, PMLR, 07–09 Jul, 2015.

[25] R. S. Sutton, D. Precup, and S. Singh, *Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning*, *Artificial intelligence* **112** (1999), no. 1-2 181–211.

[26] O. Nachum, S. S. Gu, H. Lee, and S. Levine, *Data-efficient hierarchical reinforcement learning*, *Advances in neural information processing systems* **31** (2018).

[27] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. Pieter Abbeel, and W. Zaremba, *Hindsight experience replay*, *Advances in neural information processing systems* **30** (2017).

[28] S. Cabi, S. G. Colmenarejo, A. Novikov, K. Konyushkova, S. Reed, R. Jeong, K. Zolna, Y. Aytar, D. Budden, M. Vecerik, *et. al.*, *A framework for data-driven robotics*, *arXiv preprint arXiv:1909.12200* (2019).

[29] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, *Deep reinforcement learning framework for autonomous driving*, *Electronic Imaging* **2017** (2017), no. 19 70–76.

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

[31] S. Lange, T. Gabel, and M. Riedmiller, *Batch reinforcement learning*, in *Reinforcement learning*, pp. 45–73. Springer, 2012.

[32] V. Konda and J. Tsitsiklis, *Actor-critic algorithms*, *Advances in neural information processing systems* **12** (1999).

[33] S. K. S. Ghasemipour, D. Schuurmans, and S. S. Gu, *Emaq: Expected-max q-learning operator for simple yet effective offline and online rl*, in *International Conference on Machine Learning*, pp. 3682–3691, PMLR, 2021.

[34] J. Li, Q. Vuong, S. Liu, M. Liu, K. Ciosek, H. Christensen, and H. Su, *Multi-task batch reinforcement learning with metric learning*, *Advances in Neural Information Processing Systems* **33** (2020) 6197–6210.

[35] Q. Vuong, A. Kumar, S. Levine, and Y. Chebotar, *Dasco: Dual-generator adversarial support constrained offline reinforcement learning*, in *Advances in Neural Information Processing Systems* (S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, eds.), vol. 35, pp. 38937–38949, Curran Associates, Inc., 2022.

[36] J. Wu, H. Wu, Z. Qiu, J. Wang, and M. Long, *Supported policy optimization for offline reinforcement learning*, in *Advances in Neural Information Processing Systems* (A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, eds.), 2022.

[37] A. Kumar, A. Zhou, G. Tucker, and S. Levine, *Conservative q-learning for offline reinforcement learning*, *arXiv preprint arXiv:2006.04779* (2020).

[38] C. Bai, L. Wang, Z. Yang, Z. Deng, A. Garg, P. Liu, and Z. Wang, *Pessimistic bootstrapping for uncertainty-driven offline reinforcement learning*, *arXiv preprint arXiv:2202.11566* (2022).

[39] T. Yu, G. Thomas, L. Yu, S. Ermon, J. Y. Zou, S. Levine, C. Finn, and T. Ma, *Mopo: Model-based offline policy optimization*, *Advances in Neural Information Processing Systems* **33** (2020) 14129–14142.

[40] T. Yu, A. Kumar, R. Rafailov, A. Rajeswaran, S. Levine, and C. Finn, *Combo: Conservative offline model-based policy optimization*, *Advances in Neural Information Processing Systems* **34** (2021).

[41] R. Yang, C. Bai, X. Ma, Z. Wang, C. Zhang, and L. Han, *RORL: Robust offline reinforcement learning via conservative smoothing*, in *Advances in Neural Information Processing Systems* (A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, eds.), 2022.

[42] J. Lyu, X. Ma, X. Li, and Z. Lu, *Mildly conservative q-learning for offline reinforcement learning*, in *Advances in Neural Information Processing Systems* (A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, eds.), 2022.

[43] R. Islam, P. Henderson, M. Gomrokchi, and D. Precup, *Reproducibility of benchmarked deep reinforcement learning tasks for continuous control*, *arXiv preprint arXiv:1708.04133* (2017).

[44] J. J. Callahan, *Advanced calculus: a geometric view*, vol. 1. Springer, 2010.

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

[46] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[47] T. Perkins and D. Precup, *A convergent form of approximate policy iteration*, *Advances in neural information processing systems* **15** (2002).

[48] J. Chen and N. Jiang, *Information-theoretic considerations in batch reinforcement learning*, in *International Conference on Machine Learning*, pp. 1042–1051, PMLR, 2019.

[49] S. Kakade and J. Langford, *Approximately optimal approximate reinforcement learning*, in *In Proc. 19th International Conference on Machine Learning*, Citeseer, 2002.

[50] S. M. Kakade, *On the sample complexity of reinforcement learning.* University of London, University College London (United Kingdom), 2003.

[51] K. Ciosek, Q. Vuong, R. Loftin, and K. Hofmann, *Better exploration with optimistic actor-critic*, 2019.

[52] I. Kostrikov, A. Nair, and S. Levine, *Offline reinforcement learning with implicit q-learning*, *arXiv preprint arXiv:2110.06169* (2021).

[53] X. B. Peng, A. Kumar, G. Zhang, and S. Levine, *Advantage-weighted regression: Simple and scalable off-policy reinforcement learning*, *arXiv preprint arXiv:1910.00177* (2019).

[54] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel, *Trust region policy optimization*, *CoRR* **abs/1502.05477** (2015) [arXiv:1502.0547].

[55] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, *CoRR* **abs/1707.06347** (2017) [arXiv:1707.0634].

[56] Y. Tang, M. Valko, and R. Munos, *Taylor expansion policy optimization*, in *International Conference on Machine Learning*, pp. 9397–9406, PMLR, 2020.

[57] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, *Deterministic policy gradient algorithms*, in *International conference on machine learning*, pp. 387–395, PMLR, 2014.

[58] W. Dabney, G. Ostrovski, D. Silver, and R. Munos, *Implicit quantile networks for distributional reinforcement learning*, in *International conference on machine learning*, pp. 1096–1105, PMLR, 2018.

[59] E. Parisotto, J. Ba, and R. Salakhutdinov, *Actor-mimic: Deep multitask and transfer reinforcement learning*, *CoRR* **abs/1511.06342** (2015).

[60] L. Chen, K. Lu, A. Rajeswaran, K. Lee, A. Grover, M. Laskin, P. Abbeel, A. Srinivas, and I. Mordatch, *Decision transformer: Reinforcement learning via sequence modeling*, *Advances in neural information processing systems* **34** (2021).

[61] M. Janner, Q. Li, and S. Levine, *Offline reinforcement learning as one big sequence modeling problem*, in *Advances in Neural Information Processing Systems*, 2021.

[62] R. Berkeley, *Rlkit: Reinforcement learning framework and algorithms implemented in pytorch*, .

[63] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare, *Deep reinforcement learning at the edge of the statistical precipice*, *Advances in neural information processing systems* **34** (2021) 29304–29320.

[64] Y. Jin, Z. Yang, and Z. Wang, *Is pessimism provably efficient for offline rl?*, in *International Conference on Machine Learning*, pp. 5084–5096, PMLR, 2021.

[65] M. Yin, Y. Duan, M. Wang, and Y.-X. Wang, *Near-optimal offline reinforcement learning with linear representation: Leveraging variance information with pessimism*, *International Conference on Learning Representations* (2022).

[66] S. Zou, T. Xu, and Y. Liang, *Finite-sample analysis for sarsa with linear function approximation*, *Advances in neural information processing systems* **32** (2019).

[67] L. Li, T. J. Walsh, and M. L. Littman, *Towards a unified theory of state abstraction for mdps.*, *ISAIM* **4** (2006), no. 5 9.

[68] Y. Duan, Z. Jia, and M. Wang, *Minimax-optimal off-policy evaluation with linear function approximation*, in *International Conference on Machine Learning*, pp. 2701–2709, PMLR, 2020.

[69] M. Yin and Y.-X. Wang, *Asymptotically efficient off-policy evaluation for tabular reinforcement learning*, in *International Conference on Artificial Intelligence and Statistics*, pp. 3948–3958, PMLR, 2020.

[70] R. Zhang, X. Zhang, C. Ni, and M. Wang, *Off-policy fitted q-evaluation with differentiable function approximators: Z-estimation and inference theory*, *International Conference on Machine Learning* (2022).

[71] D. Ernst, P. Geurts, and L. Wehenkel, *Tree-based batch mode reinforcement learning*, *Journal of Machine Learning Research* **6** (2005) 503–556.

[72] H. Le, C. Voloshin, and Y. Yue, *Batch policy learning under constraints*, in *International Conference on Machine Learning*, pp. 3703–3712, PMLR, 2019.

[73] S. Fujimoto, D. Meger, D. Precup, O. Nachum, and S. S. Gu, *Why should i trust you, bellman? the bellman error is a poor replacement for value error*, *arXiv preprint arXiv:2201.12417* (2022).

[74] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et. al.*, *Human-level control through deep reinforcement learning*, *nature* **518** (2015), no. 7540 529–533.

[75] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor*, *CoRR* **abs/1801.01290** (2018) [arXiv:1801.0129].

[76] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, *Continuous control with deep reinforcement learning*, *arXiv preprint arXiv:1509.02971* (2015).

[77] S. Fujimoto, H. van Hoof, and D. Meger, *Addressing function approximation error in actor-critic methods*, *CoRR* **abs/1802.09477** (2018) [arXiv:1802.0947].

[78] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, *arXiv preprint arXiv:1412.6980* (2014).

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

[80] M. I. Jordan and R. A. Jacobs, *Hierarchical mixtures of experts and the em algorithm*, *Neural computation* **6** (1994), no. 2 181–214.

[81] L. Xu, M. Jordan, and G. E. Hinton, *An alternative model for mixtures of experts*, *Advances in neural information processing systems* **7** (1994).

[82] C. Jin, Y. Zhang, S. Balakrishnan, M. J. Wainwright, and M. I. Jordan, *Local maxima in the likelihood of gaussian mixture models: Structural results and algorithmic consequences*, *Advances in neural information processing systems* **29** (2016).

[83] C. M. Bishop and M. Svensén, *Bayesian hierarchical mixtures of experts*, *arXiv preprint arXiv:1212.2447* (2012).

[84] X. Ma, L. Xia, Z. Zhou, J. Yang, and Q. Zhao, *Dsac: Distributional soft actor critic for risk-sensitive reinforcement learning*, *arXiv preprint arXiv:2004.14547* (2020).

[85] A. Müller, *Integral probability metrics and their generating classes of functions*, *Advances in Applied Probability* **29** (1997), no. 2 429–443.

[86] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, *Distributional reinforcement learning with quantile regression*, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[87] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et. al.*, *Pytorch: An imperative style, high-performance deep learning library*, *Advances in neural information processing systems* **32** (2019).

[88] M. Ahn, A. Brohan, N. Brown, Y. Chebotar, O. Cortes, B. David, C. Finn, C. Fu, K. Gopalakrishnan, K. Hausman, A. Herzog, D. Ho, J. Hsu, J. Ibarz, B. Ichter, A. Irpan, E. Jang, R. J. Ruano, K. Jeffrey, S. Jesmonth, N. J. Joshi, R. Julian, D. Kalashnikov, Y. Kuang, K.-H. Lee, S. Levine, Y. Lu, L. Luu, C. Parada, P. Pastor, J. Quiambao, K. Rao, J. Rettinghouse, D. Reyes, P. Sermanet,

N. Sievers, C. Tan, A. Toshev, V. Vanhoucke, F. Xia, T. Xiao, P. Xu, S. Xu, M. Yan, and A. Zeng, *Do as i can, not as i say: Grounding language in robotic affordances*, 2022.

[89] O. Mees, L. Hermann, and W. Burgard, *What matters in language conditioned robotic imitation learning over unstructured data*, 2022.

[90] C. Lynch and P. Sermanet, *Grounding language in play*, *CoRR* **abs/2005.07648** (2020) [arXiv:2005.0764].

[91] Y. Jiang, S. Gu, K. Murphy, and C. Finn, *Language as an Abstraction for Hierarchical Deep Reinforcement Learning*. Curran Associates Inc., Red Hook, NY, USA, 2019.

[92] C. Colas, A. Akakzia, P.-Y. Oudeyer, M. Chetouani, and O. Sigaud, *Language-conditioned goal generation: a new approach to language grounding for rl*, *LAREL Workshop 2020* **abs/2006.07043** (2020).

[93] M. Janner, Y. Du, J. B. Tenenbaum, and S. Levine, *Planning with diffusion for flexible behavior synthesis*, 2022.

[94] W. Huang, P. Abbeel, D. Pathak, and I. Mordatch, *Language models as zero-shot planners: Extracting actionable knowledge for embodied agents*, *ICML* (2022).

[95] Y. Jiang, A. Gupta, Z. Zhang, G. Wang, Y. Dou, Y. Chen, L. Fei-Fei, A. Anandkumar, Y. Zhu, and L. Fan, *Vima: General robot manipulation with multimodal prompts*, *arXiv preprint arXiv:2210.03094* (2022).

[96] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, *Code as policies: Language model programs for embodied control*, *arXiv preprint arXiv:2209.07753* (2022).

[97] W. Huang, F. Xia, T. Xiao, H. Chan, J. Liang, P. Florence, A. Zeng, J. Tompson, I. Mordatch, Y. Chebotar, *et. al.*, *Inner monologue: Embodied reasoning through planning with language models*, *arXiv preprint arXiv:2207.05608* (2022).

[98] A. Q. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. Mcgrew, I. Sutskever, and M. Chen, *GLIDE: Towards photorealistic image generation and editing with text-guided diffusion models*, in *Proceedings of the 39th International Conference on Machine Learning* (K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvari, G. Niu, and S. Sabato, eds.), vol. 162 of *Proceedings of Machine Learning Research*, pp. 16784–16804, PMLR, 17–23 Jul, 2022.

[99] J. Ho, A. Jain, and P. Abbeel, *Denoising diffusion probabilistic models*, *arXiv preprint arxiv:2006.11239* (2020).

[100] J. Ho, T. Salimans, A. Gritsenko, W. Chan, M. Norouzi, and D. J. Fleet, *Video diffusion models*, 2022.

[101] Z. Wang, J. J. Hunt, and M. Zhou, *Diffusion policies as an expressive policy class for offline reinforcement learning*, 2022.

[102] J. Ho, W. Chan, C. Saharia, J. Whang, R. Gao, A. Gritsenko, D. P. Kingma, B. Poole, M. Norouzi, D. J. Fleet, *et. al.*, *Imagen video: High definition video generation with diffusion models*, *arXiv preprint arXiv:2210.02303* (2022).

[103] U. Singer, A. Polyak, T. Hayes, X. Yin, J. An, S. Zhang, Q. Hu, H. Yang, O. Ashual, O. Gafni, *et. al.*, *Make-a-video: Text-to-video generation without text-video data*, *arXiv preprint arXiv:2209.14792* (2022).

[104] A. Zhang, R. McAllister, R. Calandra, Y. Gal, and S. Levine, *Learning invariant representations for reinforcement learning without reconstruction*, *arXiv preprint arXiv:2006.10742* (2020).

[105] T. Wang, S. S. Du, A. Torralba, P. Isola, A. Zhang, and Y. Tian, *Denoised mdps: Learning world models better than the world itself*, *arXiv preprint arXiv:2206.15477* (2022).

[106] J. Song, C. Meng, and S. Ermon, *Denoising diffusion implicit models*, *arXiv preprint arXiv:2010.02502* (2020).

[107] O. Mees, L. Hermann, E. Rosete-Beas, and W. Burgard, *Calvin: A benchmark for language-conditioned policy learning for long-horizon robot manipulation tasks*, *IEEE Robotics and Automation Letters* (2022).

[108] K. Asadi, D. Misra, and M. Littman, *Lipschitz continuity in model-based reinforcement learning*, in *International Conference on Machine Learning*, pp. 264–273, PMLR, 2018.

[109] H. K. Khalil, *Nonlinear systems third edition*, 2008.

[110] F. Berkenkamp, M. Turchetta, A. Schoellig, and A. Krause, *Safe model-based reinforcement learning with stability guarantees*, *Advances in neural information processing systems* **30** (2017).

[111] O. Nachum, S. Gu, H. Lee, and S. Levine, *Near-optimal representation learning for hierarchical reinforcement learning*, in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, OpenReview.net, 2019.

[112] L. Li, T. Walsh, and M. Littman, *Towards a unified theory of state abstraction for mdps.*, 01, 2006.

[113] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, *Exploring the limits of transfer learning with a unified text-to-text transformer*, *The Journal of Machine Learning Research* **21** (2020), no. 1 5485–5551.

[114] C. Lynch and P. Sermanet, *Language conditioned imitation learning over unstructured data*, *arXiv preprint arXiv:2005.07648* (2020).

[115] K. M. Hermann, F. Hill, S. Green, F. Wang, R. Faulkner, H. Soyer, D. Szepesvari, W. M. Czarnecki, M. Jaderberg, D. Teplyashin, *et. al.*, *Grounded language learning in a simulated 3D world*, *arXiv preprint arXiv:1706.06551* (2017).

[116] D. Bahdanau, F. Hill, J. Leike, E. Hughes, A. Hosseini, P. Kohli, and E. Grefenstette, *Learning to understand goal specifications by modelling reward*, *arXiv preprint arXiv:1806.01946* (2018).

[117] F. Hill, A. Lampinen, R. Schneider, S. Clark, M. Botvinick, J. L. McClelland, and A. Santoro, *Emergent systematic generalization in a situated agent*, 2019.

[118] H. Wang, Q. Wu, and C. Shen, *Soft expert reward learning for vision-and-language navigation*, in *European Conference on Computer Vision*, pp. 126–141, Springer, 2020.

[119] J. Luketina, N. Nardelli, G. Farquhar, J. Foerster, J. Andreas, E. Grefenstette, S. Whiteson, and T. Rocktäschel, *A survey of reinforcement learning informed by natural language*, *arXiv preprint arXiv:1906.03926* (2019).

[120] V. Zhong, T. Rocktäschel, and E. Grefenstette, *Rtfm: Generalising to novel environment dynamics via reading*, *arXiv preprint arXiv:1910.08210* (2019).

[121] X. Wang, Q. Huang, A. Celikyilmaz, J. Gao, D. Shen, Y.-F. Wang, W. Y. Wang, and L. Zhang, *Vision-language navigation policy learning and adaptation*, *IEEE transactions on pattern analysis and machine intelligence* **43** (2020), no. 12 4205–4216.

[122] K. Zheng, X. Chen, O. C. Jenkins, and X. E. Wang, *Vlmbench: A compositional benchmark for vision-and-language manipulation*, *arXiv preprint arXiv:2206.08522* (2022).

[123] J. Duan, S. Yu, H. L. Tan, H. Zhu, and C. Tan, *A survey of embodied ai: From simulators to research tasks*, *IEEE Transactions on Emerging Topics in Computational Intelligence* **6** (2022), no. 2 230–244.

[124] W. V. O. Quine, *Word and object mit press*, *Cambridge MA* (1960).

[125] H. Hu, D. Yarats, Q. Gong, Y. Tian, and M. Lewis, *Hierarchical decision making by generating and following natural language instructions*, *Advances in neural information processing systems* **32** (2019).

[126] A. W. Hanjie, V. Y. Zhong, and K. Narasimhan, *Grounding language to entities and dynamics for generalization in reinforcement learning*, in *International Conference on Machine Learning*, pp. 4051–4062, PMLR, 2021.

[127] V. Zhong, J. Mu, L. Zettlemoyer, E. Grefenstette, and T. Rocktäschel, *Improving policy learning via language dynamics distillation*, *arXiv preprint arXiv:2210.00066* (2022).

[128] J. Mu, V. Zhong, R. Raileanu, M. Jiang, N. Goodman, T. Rocktäschel, and E. Grefenstette, *Improving intrinsic exploration with language abstractions*, *arXiv preprint arXiv:2202.08938* (2022).

[129] E. Jang, A. Irpan, M. Khansari, D. Kappler, F. Ebert, C. Lynch, S. Levine, and C. Finn, *Bc-z: Zero-shot task generalization with robotic imitation learning*, in *Conference on Robot Learning*, pp. 991–1002, PMLR, 2022.

[130] E. Perez, F. Strub, H. De Vries, V. Dumoulin, and A. Courville, *Film: Visual reasoning with a general conditioning layer*, in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[131] A. Brohan, N. Brown, J. Carbajal, Y. Chebotar, J. Dabis, C. Finn, K. Gopalakrishnan, K. Hausman, A. Herzog, J. Hsu, *et. al.*, *Rt-1: Robotics transformer for real-world control at scale*, *arXiv preprint arXiv:2212.06817* (2022).

[132] A. Jaegle, S. Borgeaud, J.-B. Alayrac, C. Doersch, C. Ionescu, D. Ding, S. Koppula, D. Zoran, A. Brock, E. Shelhamer, *et. al.*, *Perceiver io: A general architecture for structured inputs & outputs*, *arXiv preprint arXiv:2107.14795* (2021).

[133] S. Reed, K. Zolna, E. Parisotto, S. G. Colmenarejo, A. Novikov, G. Barth-Maron, M. Gimenez, Y. Sulsky, J. Kay, J. T. Springenberg, *et. al.*, *A generalist agent*, *arXiv preprint arXiv:2205.06175* (2022).

[134] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, *Deep unsupervised learning using nonequilibrium thermodynamics*, in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of *Proceedings of Machine Learning Research*, (Lille, France), pp. 2256–2265, PMLR, 07–09 Jul, 2015.

[135] J. Ho, C. Saharia, W. Chan, D. J. Fleet, M. Norouzi, and T. Salimans, *Cascaded diffusion models for high fidelity image generation*, *Journal of Machine Learning Research* **23** (2022), no. 47 1–33.

[136] X. L. Li, J. Thickstun, I. Gulrajani, P. Liang, and T. B. Hashimoto, *Diffusion-lm improves controllable text generation*, 2022.

[137] K. Rasul, C. Seward, I. Schuster, and R. Vollgraf, *Autoregressive denoising diffusion models for multivariate probabilistic time series forecasting*, .

[138] Y. Dai, M. Yang, B. Dai, H. Dai, O. Nachum, J. Tenenbaum, D. Schuurmans, and P. Abbeel, *Learning universal policies via text-guided video generation*, *arXiv preprint arXiv:2302.00111* (2023).

[139] C. Chi, S. Feng, Y. Du, Z. Xu, E. Cousineau, B. Burchfiel, and S. Song, *Diffusion policy: Visuomotor policy learning via action diffusion*, *arXiv preprint arXiv:2303.04137* (2023).

[140] A. Ajay, Y. Du, A. Gupta, J. Tenenbaum, T. Jaakkola, and P. Agrawal, *Is conditional generative modeling all you need for decision-making?*, *arXiv preprint arXiv:2211.15657* (2022).

[141] L. Weng, *What are diffusion models?*, *lilianweng.github.io* (Jul, 2021).

[142] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, *Deep unsupervised learning using nonequilibrium thermodynamics*, in *International Conference on Machine Learning*, pp. 2256–2265, PMLR, 2015.

[143] M. Welling and Y. W. Teh, *Bayesian learning via stochastic gradient langevin dynamics*, 2011.

[144] Y. Song, J. Sohl-Dickstein, D. P. Kingma, A. Kumar, S. Ermon, and B. Poole, *Score-based generative modeling through stochastic differential equations*, *arXiv preprint arXiv:2011.13456* (2020).

[145] E. Strubell, A. Ganesh, and A. McCallum, *Energy and policy considerations for deep learning in NLP*, in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, (Florence, Italy), pp. 3645–3650, Association for Computational Linguistics, July, 2019.

[146] A. S. Luccioni, S. Viguier, and A.-L. Ligozat, *Estimating the carbon footprint of bloom, a 176b parameter language model*, *arXiv preprint arXiv:2211.02001* (2022).