Unified Movement Primitives and Advanced Reinforcement Learning for Efficient Robotic Skill Learning

Zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

von der KIT-Fakultät für Informatik des
Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Ge Li

geb. in Peking, China

Tag der mündlichen Prüfung: 20.11.2025
1. Referent: Prof. Dr. Gerhard Neumann
2. Referent: Dr. Freek Stulp (DLR)

Abstract

Artificial intelligence has profoundly transformed how machines interact with the world, enabling physical agents such as robots to become intelligent and autonomous. Machine learning techniques, including imitation learning and reinforcement learning (RL), allow robots to perceive, reason, and act based on data or predefined reward signals. A successful control policy serves as the robot's brain, determining when and how it acts. In most approaches, actions are predicted at every time step, leading to extremely high decision frequencies.

In contrast, humans—whether as infants or experts—do not make decisions at such high rates, nor consciously think about every micro-movement at very high frequency. Instead, we *just do it*: moving fingers to grasp an object, swinging an arm to hit a tennis ball, or shooting a football using our legs. We decide on a course of action, execute a sequence of movements, and make slight adjustments during execution.

Movement Primitives (MPs) are motion representation and generation techniques inspired by biological movements. They encode and modulate complex motions using latent parameters, typically of lower dimensionality than the original trajectory. MPs are widely used in robot learning and, more recently, have been integrated with deep learning architectures to address issues caused by per-step action prediction—such as poor smoothness, abrupt transitions, and inefficient exploration in reinforcement learning.

Despite their advantages, MPs face two major challenges. First is *representation capacity*: the ability to capture both the dynamical and statistical characteristics of movement, ensuring smooth action replanning while modeling motion correlations and uncertainty. Second is *integration efficiency*: combining MPs effectively with reinforcement learning so that robots do not sample random actions at each time step, resulting in behaviors resembling a random walk.

This dissertation addresses these challenges and presents several algorithms bridging the gap. First, it unifies two classical MP models—Dynamic Movement Primitives (DMPs) and Probabilistic Movement Primitives (ProMPs)—by analytically solving the underlying ODEs of DMPs. The resulting framework, *ProDMPs*, inherits the strengths of both, capturing dynamical smoothness and probabilistic structure. Next, two novel RL methods are proposed: *Temporally Correlated Episodic RL* and *Transformer-based Episodic RL*, which enable fine-grained updates of MP-based policies and precise evaluation of action sequence values.

This dissertation presents my research on action sequence modeling and policy learning, conducted during my PhD studies. It demonstrates how I improved movement primitive models, thereby enhancing learning efficiency and policy performance for robotic agents.

Zusammenfassung

Künstliche Intelligenz hat die Art und Weise, wie Maschinen mit der Welt interagieren, grundlegend verändert und es physischen Agenten wie Robotern ermöglicht, intelligent und autonom zu agieren. Machine Learning-Verfahren, einschließlich Imitation Learning und Reinforcement Learning (RL), befähigen Roboter, auf Grundlage von Daten oder vordefinierten Belohnungssignalen zu erkennen, zu schlussfolgern und zu handeln. Eine erfolgreiche Policy fungiert als „Gehirn“ des Roboters und bestimmt, wann und wie er handelt. In den meisten Ansätzen werden Actions in jedem Zeitschritt vorhergesagt, was zu extrem hohen Entscheidungsfrequenzen führt.

Im Gegensatz dazu treffen Menschen – ob als Kleinkinder oder Experten – Entscheidungen nicht in solch hoher Frequenz und denken auch nicht bewusst über jede Mikrobewegung bei sehr hohen Frequenzen nach. Stattdessen *tun wir es einfach*: Wir bewegen die Finger, um ein Objekt zu greifen, schwingen den Arm oder schießen einen Fußball. Wir entscheiden uns für eine Action Sequence, führen diese aus und nehmen während der Ausführung nur kleine Anpassungen vor.

Movement Primitives (MPs) sind Bewegungsdarstellungs- und -generierungstechniken, die von biologischen Bewegungen inspiriert sind. Sie kodieren und modulieren komplexe Bewegungen mithilfe latenter Parameter, die typischerweise eine geringere Dimensionalität als die ursprüngliche Trajektorie aufweisen. MPs werden in der Robotik breit eingesetzt und wurden in jüngerer Zeit mit Deep Learning-Architekturen kombiniert, um Probleme zu adressieren, die durch schrittweise Action Prediction entstehen – wie geringe Glattheit, abrupte Übergänge und ineffiziente Exploration im Reinforcement Learning.

Trotz ihrer Vorteile stehen MPs vor zwei zentralen Herausforderungen. Erstens die *Representation Capacity*: die Fähigkeit, sowohl die dynamischen als auch die statistischen Eigenschaften einer Bewegung zu erfassen, um reibungslose Action Replanning zu gewährleisten und gleichzeitig Bewegungskorrelationen und Unsicherheiten zu modellieren. Zweitens die *Integration Efficiency*: die effektive Kombination von MPs mit Reinforcement Learning, sodass Roboter nicht bei jedem Zeitschritt zufällige Actions wählen und sich wie in einem Random Walk verhalten.

Diese Dissertation befasst sich mit diesen Herausforderungen und präsentiert mehrere Algorithmen, die diese Lücke schließen. Zunächst werden zwei klassische MP-Modelle – Dynamic Movement Primitives (DMPs) und Probabilistic Movement Primitives (ProMPs) – durch analytische Lösung der zugrunde liegenden Differentialgleichungen der DMPs vereinigt. Das resultierende Framework, *ProDMPs*, vereint die Stärken beider Modelle und erfasst sowohl die dynamische Glattheit als auch die probabilistische Struktur. Darüber hinaus werden zwei neue RL-Methoden vorgestellt: *Temporally Correlated Episodic RL* und *Transformer-based Episodic RL*, die feingranulare Aktualisierungen von MP-basierten Policies und eine präzise Bewertung von Action Sequences ermöglichen.

Diese Dissertation präsentiert meine Forschung zur Action Sequence Modeling und zum Policy Learning, die ich im Rahmen meiner Promotion durchgeführt habe. Sie zeigt, wie ich Movement Primitive-Modelle verbessert habe und dadurch die Learning Efficiency sowie die Policy Performance von Robotik-Agenten gesteigert habe.

Acknowledgements

Firstly, I would like to sincerely express my gratitude to my PhD advisor, **Geri**, for offering me the invaluable opportunity to be his doctoral student, to learn from him, and to contribute to the fields of robotics and machine learning. I truly believe that Geri provides the best research atmosphere and the fairest professor–student relationship I could have ever imagined before starting this journey. For me, this opportunity has been extremely valuable—it has significantly changed the trajectory of my life and influenced my entire career path. Until today, I still clearly remember the excitement and appreciation I felt when I received Geri's offer. Over the past five years, I have gained immense knowledge and skills through our discussions and his lectures. Beyond that, his calm and humble character has deeply shaped how I perceive and respond to challenges in both work and life. There are so many admirable qualities about Geri that I could write a whole storybook about him. Among them, the most remarkable is his mysterious yet insightful intuition—he always manages to offer precise and helpful suggestions, and his perspectives consistently prove to be correct. After five years of working with him, I have grown from a naive student into a confident machine learner and roboticist. Thank you, Geri!

Next, I would like to thank **Rudi** for becoming my second PhD mentor since 2022. Working with him and his team has broadened my horizons and fostered collaborations in various areas of robotic research, especially imitation learning and diffusion models. Beyond the technical aspects, I have learned many valuable skills from him in presentation and academic writing. He taught me that research is not only about building new wheels, but also about communicating with other researchers, exchanging ideas, and speaking out so that our work is recognized and can benefit the community. Another important lesson I learned from him is to never give up hope. In my ICLR 2024 submission, we turned a review score of two rejects and two weak rejects into four weak accepts, eventually securing acceptance. This experience left me with deep respect for Rudi's persistence and optimism. If Geri's character can be described as water—calm yet full of wisdom—then Rudi's must be fire—outgoing and energetic. Working with both of them has made everything much easier. I feel truly fortunate and supported to have them in my PhD journey.

Apart from the amazing professors, I have also spent five years with two strong research teams and fantastic members. **Onur** offered me tremendous support when I started my PhD. His kind personality made all our collaborations smooth, and he has always been helpful and supportive—not only to me but to everyone in the lab. **Hongyi** and I have shared both an office and a research focus for about four years. Even at the beginning, when he did his Praktikum and master's thesis with me, he had already demonstrated great responsibility and a strong work ethic in research. Later, he became a PhD student, and our collaboration grew even closer. Interestingly, from that point on,

every paper we submitted together was accepted. This was certainly not a coincidence, but the result of our seamless collaboration, constant exchange of ideas, shared ownership, and our willingness to help and support each other—ingredients that make success possible. **David and Xinkai**, two cool guys from my home country, work extremely hard in the lab. I am fortunate to have joined their research projects and to have learned many skills beyond my own research focus—ranging from various imitation learning architectures to VR/AR frameworks. These experiences broadened my perspective on robotics research and strengthened my motivation in this field. Because we share a similar taste in food, we often went for lunch or dinner together, which always made me feel relaxed, especially during tight deadlines. **Fabian Otto, Michael, and Vaisakh**, three research experts, collaborated closely with me in the early stage of my PhD. Their outstanding research work significantly supported my own projects. I gained a wealth of knowledge, skills, and insights from them—ranging from designing architectures, tuning hyperparameters, to training models. Their support helped me navigate the most challenging phase of my PhD and played a key role in shaping my research direction. **Max Hüttenrauch, Philipp Becker, and Denis Blessing** are three colleagues I worked with as a teaching assistant. I appreciate their patience and dedication in preparing lectures, homework, and exams together. Their efforts ensured the great success and popularity of Geri's lectures, and this teaching assistant experience gave me a unique sense of accomplishment during my PhD. **Max Li**, an amazing and supportive teammate who implemented *ClusterWork2*, has made my life much easier every time I submit experiments. I am always grateful for this smart tool, and it has inspired me to develop my own tools to benefit others. **Christine**, the kindest, most supportive, and gentle person at KIT, handled countless administrative tasks for our lab and managed an endless number of my documents. I have never met anyone as multi-talented as she is, and I always tell my colleagues that having Christine as the secretary for both of our labs is our greatest fortune. **Aleks and Tai**, the two RL experts and true gentlemen in our lab, kindly supported me by sharing their valuable computational resources during my ICLR 2025 submission. They are the unsung heroes who contributed to my publication and helped ensure that I could complete my PhD smoothly. I deeply appreciate their generosity. I also thank **Philipp Dahlinger, Nic, Niklas, Balázs, Emiliyan, Enes, Huy, Roman, Zeqi, Max Nagy, Serge, Dominik, Xi, Weiran, Qian, Florian, Moritz, Paul, Nils, and Pankhuri** for creating such a pleasant working atmosphere and for the great times we shared in the lab.

I am deeply grateful to **my parents**, who sent me to Germany for my education and have supported me throughout this journey. During my PhD, due to the Covid-19 pandemic and my demanding schedule, I only met them once during my ICLR trip in Singapore. Over these years, I have gradually come to understand the challenges an adult faces in life and society, and I appreciate even more the warm and safe environment they created for my childhood. It was they who built the foundation of my character and shaped me into a responsible and honest person.

The greatest fortune of my PhD journey was meeting my wife **Juan**, my dearest love. She not only supported me unconditionally at home, sharing my feelings—especially during moments of extreme stress and frustration—but also stood by me during major deadlines. It was she who strengthened my resolve to work harder and keep pushing the quality of my research. Although her name never

appeared in any of my publications, her contributions have earned her the title of my *Support Captain*.

Lastly, I would like to thank Freek for agreeing to be my second referee, and I would like to thank the PhD committee for serving as my examiners.

Contents

Table 1.: List of English math letters used in this dissertation.

Symbol	Description
$a, \mathcal{A}$	RL actions and action space
A	RL advantage function
b	The time step of the initial conditions
$\mathcal{B}$	Replay Buffer
c	Coefficients of the conventional solutions of DMPs' ODE
$d,\ D$	Index and the amount of DoFs
f	Forcing function of the original DMPs
F	The source term in the DMPs' non-homogeneous ODEs
g	The Goal term of DMPs and ProDMPs
G	RL return
$\boldsymbol{H}$	A combination of ProDMPs' basis and boundary conditions terms, used in distribution computation
i	index used in observation and learning objective of MDP
I	Integral terms used in DMPs' ODE solution derivation
$\boldsymbol{I}$	Identity matrix
$j,\ J$	index of the product of importance sampling in N-step return
$\mathcal{J}$	The objective function to be maximized
k, K	Index and the amount of the trajectory segments
I	The two indefinite integrals in the closed form solution of DMPs' ODE
L	Length of a trajectory segment
$\boldsymbol{L}$	Cholesky Decomposition of Covariance Matrix
l	index of future time step in policy gradient
$\mathcal{L}$	The loss function to be minimized
m, M	Index and number of basis functions and weights per DoFs
n, N	Index and the amount of the steps for multi-step return
$\mathcal{N}$	Normal (Gaussian) Distribution
$\boldsymbol{o},\ \mathcal{O}$	Sensory observation and a set of observations
$p, \mathcal{P}$	probability, probability space
p, q	intermediate functions that formulate ProDMPs
Q	The state-action value function
$r, \mathcal{R}$	reward function and reward space
$s, \mathcal{S}$	state and state space
t, T	Index of time steps and the length of trajectory, i.e. the number of total time steps
$\mathcal{T}$	State transition probability
$\mathcal{U}$	Uniform Distribution
V	The state Value function
$\boldsymbol{w}, \boldsymbol{w_g}$	The weights of DMPs, ProMPs, and ProDMPs. A concatenation of weights and goal
x	The phase of the canonical system, which decays exponentially w.r.t time
$y,\ \dot{y},\ \ddot{y}$	Trajectory position, velocity, and acceleration of a single time step and a single DoF
y	target value for off-policy RL critic updates
Y	Position trajectory of a single time step and multiple DoF
$\boldsymbol{z}$	Latent variable

Table 2.: List of Greek math letters used in this dissertation.

Symbol	Description
α_x	Exponential decaying coefficient of the canonical system
α, β	Spring-damper coefficient of DMPs
Δ	The characteristic equation of DMPs' homogeneous ODEs
ϵ	Gaussian white noise
θ	Policy Networks' learnable parameters
$\mu, \boldsymbol{\mu}$	The mean of a certain probabilistic distribution
ξ, $\boldsymbol{\xi}$	One or multi time-step term determined by boundary conditions in ProDMPs' trajectory computation
σ^2	The variance of a certain probabilistic distribution
$\boldsymbol{\Sigma}$	The covariance of a certain probabilistic distribution
τ	Time scaling factor of DMPs, determining the execution speed; State-action trajectory of RL
ρ	importance sampling ratio
π	Policy
γ	Discount Factor
ϕ	Critic Networks' learnable parameters
φ	Basis function forming up DMPs' forcing term
$\boldsymbol{\Phi}$	Basis functions of one DoF used by ProMPs and ProDMPs, written in matrix
$\boldsymbol{\Psi}$	Blocked diagonal basis function matrix used by ProMPs and ProDMPs for multi DoFs trajectory
$\dot{\boldsymbol{\Phi}}, \dot{\boldsymbol{\Psi}}$	Single and blocked diagonal velocity basis function matrices
Ω	General reward signals in policy gradient

1. Introduction

1.1. Research Questions Addressed by this Dissertation

Movement Primitives (MPs) [1, 2, 3, 4] are parameterized techniques that enable the compact representation and efficient generation of robotic trajectories. Because of their inherent modularity and adaptability, MPs have been applied successfully to a wide range of robotic skill-acquisition tasks [5, 6], where they provide structured trajectory generators that reduce complex movement-learning problems to the tuning of a comparatively small set of parameters.

MPs are commonly divided into two broad categories. *Dynamical-systems* approaches [1, 2] emphasize smooth, stable, and adaptive motion generation by explicitly modelling the underlying differential equations of movement. *Probabilistic* approaches [3, 4] instead capture trajectory statistics and correlations, thereby enabling principled treatment of uncertainty, variability, and conditioning on contextual variables. However, most existing MP frameworks focus on only *one* of these aspects—either dynamical smoothness or probabilistic expressiveness—thereby limiting their representational power and versatility in practical applications.

Integrating MPs with reinforcement learning (RL) adds an additional layer of complexity owing to the compact yet highly structured nature of MP parameterizations. RL algorithms can be broadly categorized into *step-based RL* [7, 8], in which the agent selects an action at every time step, and *episodic RL* (ERL) [9, 10], in which the agent predicts an entire sequence of actions—often through an MP—before executing it. Step-based RL takes advantage of fine-grained state–action–reward information, yet it frequently produces discontinuous or jerky trajectories because temporal correlations between successive actions are captured only implicitly. Conversely, episodic RL yields temporally coherent motion through its holistic sequence-prediction paradigm, but it often learns inefficiently because each trajectory is treated as an indivisible unit, thereby discarding informative step-level feedback that could accelerate credit assignment.

A further complication arises from the widespread reliance of episodic RL methods on *on-policy* learning rules. On-policy algorithms must collect fresh data for every policy update and are unable to exploit past experience effectively. This restriction leads to high data requirements and slow convergence, both of which are particularly detrimental in robotics scenarios that demand rapid adaptation, safe exploration, and efficient use of limited interaction budgets.

These open challenges motivate the research conducted in this dissertation. First, I aim to develop a unified MP framework that combines the smoothness and stability of dynamical-systems models with the uncertainty-aware expressiveness of probabilistic models. Second, I investigate algorithms

that incorporate detailed step-based information into episodic RL without sacrificing the temporal coherence characteristic of sequence-level planning. Third, I explore the use of advanced sequence-processing architectures—most notably Transformers [11]—to enable data-efficient *off-policy* updates within ERL, thereby enhancing both representational capability and learning efficiency for trajectory-based robotic policies.

This dissertation addresses the following **research questions**:

- **[RQ1]** How can dynamical smoothness and probabilistic trajectory modelling be effectively unified within a single MP framework?
- **[RQ2]** How can fine-grained step-based information be integrated into episodic reinforcement-learning frameworks to improve learning efficiency without sacrificing trajectory coherence?
- **[RQ3]** How can modern sequence architectures—particularly Transformers—facilitate data-efficient, off-policy updates in episodic RL when tackling complex robotic tasks?

1.2. Structure of the Dissertation

Chapter 2 surveys the foundational material required for this work, including Movement Primitives, on- and off-policy reinforcement learning, Bayesian aggregation, Transformer architectures, and Gaussian policy representations. Chapter 3 presents a method that unifies probabilistic and dynamical movement primitives, thereby addressing RQ1. Chapter 4 bridges the gap between step-based RL and episodic RL by combining episodic exploration with per-step policy updates, thus addressing RQ2. Chapter 5 introduces an off-policy actor–critic algorithm that employs Transformer-based critics within the ERL paradigm, thereby addressing RQ3. Finally, Chapter 6 summarizes the main contributions of the dissertation and outlines promising directions for future research.

1.3. Research Contribution

The following three sections are reprints of the abstracts of the respective publications this dissertation is based on. Those publications are discussed in Sections 3, 4 and 5 in more detail.

1.3.1. Unifying Dynamical and Probabilistic Representations in Movement Primitives

Based on *"ProDMP: A Unified Perspective on Dynamic and Probabilistic Movement Primitives", [12], published in the IEEE Robotics and Automation Letters (RAL), 2023.*

Movement Primitives are a well-known concept to represent and generate modular trajectories. MPs can be broadly categorized into two types: a) dynamics-based approaches that generate

smooth trajectories from any initial state, e. g., Dynamic Movement Primitives, and b) probabilistic approaches that capture higher-order statistics of the motion, e. g., Probabilistic Movement Primitives. To date, however, there is no MP method that unifies both, i. e. that can generate smooth trajectories from an arbitrary initial state while capturing higher-order statistics. In this paper, we introduce a unified perspective of both approaches by solving the ODE underlying the DMPs. We convert expensive online numerical integration of DMPs into position and velocity basis functions that can be used to represent trajectories or trajectory distributions similar to ProMPs while maintaining all the properties of dynamical systems. Since we inherit the properties of both methodologies, we call our proposed model Probabilistic Dynamic Movement Primitives. Additionally, we embed ProDMPs in deep neural network architecture and propose a new cost function for efficient end-to-end learning of higher-order trajectory statistics. To this end, we leverage Bayesian Aggregation for non-linear iterative conditioning on sensory inputs. Our proposed model achieves smooth trajectory generation, goal-attractor convergence, correlation analysis, non-linear conditioning, and online replanning in one framework.

1.3.2. Enhancing Efficiency in Episodic Reinforcement Learning through Step-based Policy Updates

Based on *"Open the Black Box: Step-Based Policy Updates for Temporally-Correlated Episodic Reinforcement Learning," [13], published at the International Conference on Learning Representations, ICLR, 2024.*

Current advancements in reinforcement learning (RL) have predominantly focused on learning step-based policies that generate actions for each perceived state. While these methods efficiently leverage step information from environmental interaction, they often ignore the temporal correlation between actions, resulting in inefficient exploration and unsmooth trajectories that are challenging to implement on real hardware. Episodic RL (ERL) seeks to overcome these challenges by exploring in parameters space that capture the correlation of actions. However, these approaches typically compromise data efficiency, as they treat trajectories as opaque *black boxes*. In this work, we introduce a novel ERL algorithm, Temporally-Correlated Episodic RL (TCE), which effectively utilizes step information in episodic policy updates, opening the 'black box' in existing ERL methods while retaining the smooth and consistent exploration in parameter space. TCE synergistically combines the advantages of step-based and episodic RL, achieving comparable performance to recent ERL methods while maintaining data efficiency akin to state-of-the-art (SoTA) step-based RL.

1.3.3. Transformer-based Critic Updates for Off-Policy Episodic Reinforcement Learning

Based on *"TOP-ERL: Transformer-based Off-Policy Episodic Reinforcement Learning,", [14], published at the International Conference on Learning Representations (ICLR), 2025.*

This work introduces Transformer-based Off-Policy Episodic Reinforcement Learning (TOP-ERL), a novel algorithm that enables off-policy updates in an ERL framework. In ERL, policies predict entire action trajectories over multiple time steps instead of single per-step actions. These trajectories are typically parameterized by trajectory generators such as Movement Primitives (MP), allowing for smooth and efficient exploration over long horizons while capturing temporal correlations. However, ERL methods are often constrained to on-policy frameworks due to the difficulty of evaluating state-action values for action sequences, limiting their sample efficiency and preventing the use of more efficient off-policy architectures. TOP-ERL addresses this shortcoming by segmenting long action sequences and estimating the state-action values for each segment using a transformer-based critic architecture alongside an N-step return estimation. These contributions result in efficient and stable training that is reflected in the empirical results conducted on sophisticated robot learning environments. TOP-ERL significantly outperforms state-of-the-art RL methods. Thorough ablation studies additionally show the impact of key design choices on the model performance.

2. Research Background and Prerequisites

In this chapter, we summarize the theoretical background relevant to the methods developed in this dissertation. We begin in Section 2.1 with an overview of movement primitives (MPs), focusing on parameterized trajectory representation and generation. This includes Dynamic Movement Primitives (DMPs) [1, 2], Probabilistic Movement Primitives (ProMPs) [3], Conditional Neural Movement Primitives (CNMPs) [4], and other variants [15, 16] commonly used in robot learning and closely related to the approaches discussed in this dissertation. In Section 2.2, we introduce the fundamentals of modern deep reinforcement learning (RL), with an emphasis on recent advances in continuous control. We cover both on- and off-policy RL, importance sampling techniques, and policy updates within trust regions. Additionally, we discuss two major RL paradigms used for robotic skill learning: step-based methods and episodic methods, which differ in how they predict actions and update policies. Finally, Section 2.3 briefly presents other essential techniques employed in this work, including Bayesian information aggregation [17] and Transformers [11].

2.1. Movement Primitives

We cover the major movement primitives methodologies used in the literature. For clarity, we start with a single degree of freedom (DoF) system ($D = 1$) and then expand to multi-DoF systems ($D > 1$).

2.1.1. Dynamic Movement Primitives (DMPs)

Schaal [1] described a single movement as a trajectory $\boldsymbol{y}_{0:T} = [y(0),\ y(1),\ ...,\ y(T)]$, which is governed by a second-order linear dynamical system with a non-linear forcing function f. The mathematical representation is given by [1]

$$\tau^2 \ddot{y} = \alpha(\beta(g - y) - \tau \dot{y}) + f(x), \quad f(x) = x \frac{\sum_m \varphi_m(x) w_m}{\sum_m \varphi_m(x)} = x \boldsymbol{\varphi}_x^\top \boldsymbol{w}, \tag{2.1}$$

where $y = y(t),\ \dot{y} = \mathrm{d}y/\mathrm{d}t,\ \ddot{y} = \mathrm{d}^2 y/\mathrm{d}t^2$ denote the position, velocity, and acceleration of the system at a specific time $t \in [0, T]$, respectively. Constants α and β are spring-damper parameters,

[1] Here, we present the original DMPs definition introduced in [1]. Later Ijspeert et al. [2] extended the definition of the forcing function by incorporating a $g - y_0$ term in the forcing function to further modulate the motion amplitude.

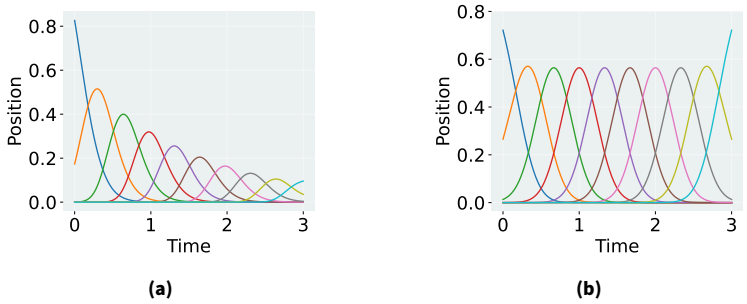


Figure 2.1.: (a) DMP's basis functions $\boldsymbol{\varphi}_x$ with exponential decaying phase x. The basis functions converge to zero when $t \to \infty$ (b) ProMP's normalized RBF basis functions Φ.

g represents the goal attractor of the system when $t \to \infty$, and τ is a time constant that modulates the speed of trajectory execution. To ensure convergence towards the goal, DMPs employ a forcing function governed by an exponentially decaying phase variable $x(t) = \exp(-\alpha_x/\tau; t)$, as shown in Fig 2.1a. In Eq. (2.1), $\varphi_i(x)$ represents the basis functions for the forcing term, typically choosing radial basis function (RBF) [18, 19] for striking movement or Von-Mises basis functions [20, 21] for periodic motions. The trajectory's shape as it approaches the goal is determined by the weight parameter vector $\boldsymbol{w} = [w_1, \ ..., \ w_M]$. The trajectory $\boldsymbol{y}_{0:T}$ is typically obtained by numerically integrating the dynamical system from $t = 0$ to $t = T$ [22, 15]. Starting from the initial position y_0 and velocity $\dot{y}_0$, *Euler integration* proceeds iteratively:

$$\begin{aligned} &\textbf{Acceleration:} && \ddot{y}_t = \frac{1}{\tau^2}[\alpha(\beta(g - y_{t-1}) - \tau\dot{y}_{t-1}) + f(x(t-1))], \\ &\textbf{Velocity:} && \dot{y}_t = \dot{y}_{t-1} + \ddot{y}_{t-1}\mathrm{d}t, \\ &\textbf{Position:} && y_t = y_{t-1} + \dot{y}_{t-1}\mathrm{d}t. \end{aligned} \tag{2.2}$$

Extend to High DoF Systems Most robotic platforms comprise multiple DoF. To generate trajectories in such settings, DMPs treat each degree of freedom separately, assigning it an independent M-dimensional weight vector $\boldsymbol{w}^d \in \mathbb{R}^{\mathbb{M}}, d = 1, ..., D$. The weights vector of all DoF $\boldsymbol{w}$ can then be represented as a concatenation of these individual vectors $\boldsymbol{w} = [{\boldsymbol{w}^1}^\top, ..., {\boldsymbol{w}^D}^\top]^\top$.

Challenges in DMPs. The numerical procedure described above is both computationally expensive and inflexible. To compute a segment that lies near the end of an episode, the system must first integrate from $t = 0$ all the way to the beginning of that segment. When DMPs are coupled with deep networks that predict the weights $\boldsymbol{w}$, the forward and backward passes of the network have to propagate through these iterative integrations in Eq. (2.2), greatly increasing the computational burden.

A second major drawback of the DMPs formulation is its limited ability to represent trajectory statistics, such as uncertainty or variance. Starting from a parameter distribution $p(\boldsymbol{w})$, obtaining

the induced trajectory distribution $p(\boldsymbol{y}_{0:T})$ is non-trivial because it requires propagating the distribution through the numerical integrator. Earlier approaches attempted to circumvent this issue by covering the trajectory space with Gaussian mixture models, yielding the so-called GMM/GMR-DMPs [23, 24]. However, these hybrids ignore temporal correlations and do not provide a proper generative model for entire trajectories. Other works inferred distributions over the weight vector alone [25, 26], but left the mapping from weights to trajectories implicit; consequently, the trajectory distribution cannot be queried directly, nor can the weight distribution be learned end-to-end from demonstrations. By contrast, methods such as ProMPs [3], which we review in the next section, model trajectory statistics explicitly and thereby overcome these limitations.

2.1.2. Probabilistic Movement Primitives (ProMPs)

Paraschos et al. [3] introduced a framework for modeling MPs using trajectory distributions that capture both temporal and inter-dimensional correlations. Unlike DMPs, which use a forcing term, ProMPs directly model the intended trajectory using a *linear basis function* formulation, as:

$$\textbf{A Single Trajectory:} \qquad \boldsymbol{y}_{0:T} = \boldsymbol{\Phi}_{0:T}^{\top} \boldsymbol{w}. \tag{2.3}$$

Here, the matrix $\boldsymbol{\Phi}_{0:T}$ is a $M \times T$ matrix, representing the basis functions for each time step $t = 0, ..., T$. Typically, ProMPs choose normalized RBF functions as basis functions, as shown in Fig 2.1b. Similar to DMPs, these basis functions can be defined in terms of a phase variable instead of time. Notably, ProMPs place the basis functions in trajectory (position) space, whereas DMPs define them in force space. Thanks to this linear basis-function structure, ProMPs can model the probability of observing a trajectory $p(\boldsymbol{y}_{0:T})$ given a weight distribution $p(\boldsymbol{w}) \sim \mathcal{N}(\boldsymbol{w} \mid \boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}})$. The resulting distribution is a linear Gaussian, where σ_y^2 is a regularization noise to ensure $\boldsymbol{\Sigma}_{\boldsymbol{y}}$ remains positive definite[2]:

$$\textbf{Trajectory Distributions:} \quad p(\boldsymbol{y}_{0:T};\ \boldsymbol{\mu}_{\boldsymbol{y}}, \boldsymbol{\Sigma}_{\boldsymbol{y}}) = \mathcal{N}(\boldsymbol{\Phi}_{0:T}^{\top} \boldsymbol{\mu}_{\boldsymbol{w}},\ \boldsymbol{\Phi}_{0:T}^{\top} \boldsymbol{\Sigma}_{\boldsymbol{w}} \boldsymbol{\Phi}_{0:T}\ + \sigma_y^2 \boldsymbol{I}). \tag{2.4}$$

ProMPs allow for flexible manipulation of MP trajectories through probabilistic operators applied to $p(\boldsymbol{w})$, such as conditioning, combination, and blending [6, 16, 27, 28, 29] and achieve variable-stiffness controllers [30, 31].

Extend to High DoF Systems Similar to DMPs, ProMPs represent multi-DoF trajectories by concatenating each DoF 's weight vector. To preserve the linear basis-function formulation in high-DoF settings, the matrix $\boldsymbol{\Phi}_{0:T}$ is repeated in a blocked-diagonal fashion, resulting in a compound basis func-

[2] The resulting distribution $p(\boldsymbol{y})$ is low rank ($\text{rank}(\boldsymbol{\Sigma}_{\boldsymbol{y}}) = \text{rank}(\boldsymbol{\Sigma}_{\boldsymbol{w}})$), so it cannot be sampled directly but is useful for likelihood evaluation.

tion matrix $\mathbf{\Psi}_{0:T}$. Given a joint distribution over the concatenated weights $\boldsymbol{w} = [{\boldsymbol{w}^1}^\top, ..., {\boldsymbol{w}^D}^\top]^\top$, ProMPs compute the multi-DoF trajectory and its distribution as:

$$\textbf{Multi-DoF Trajectory.:} \qquad \boldsymbol{Y}_{0:T} = \begin{bmatrix} \boldsymbol{y}^1_{0:T} \\ \vdots \\ \boldsymbol{y}^D_{0:T} \end{bmatrix} = \begin{bmatrix} \mathbf{\Phi}^\top_{0:T} & \cdots & \mathbf{0} \\ \vdots & \ddots & \vdots \\ \mathbf{0} & \cdots & \mathbf{\Phi}^\top_{0:T} \end{bmatrix} \boldsymbol{w} = \mathbf{\Psi}^\top_{0:T} \boldsymbol{w}, \tag{2.5}$$

$$\textbf{Multi-DoF Distribution.:} \qquad p(\boldsymbol{Y}_{0:T}; \boldsymbol{\mu}_{\boldsymbol{w}}, \mathbf{\Sigma}_{\boldsymbol{w}}) = \mathcal{N}\left(\mathbf{\Psi}^\top_{0:T}\boldsymbol{\mu}_{\boldsymbol{w}},\ \mathbf{\Psi}^\top_{0:T}\mathbf{\Sigma}_{\boldsymbol{w}}\mathbf{\Psi}_{0:T} + \sigma^2_y\mathbf{I}\right). \tag{2.6}$$

Learn weights from trajectories. Since ProMPs are using a linear basis function model, given a trajectory and basis functions, we can easily compute the weights used to generate such a trajectory using *ridge regression* [32], with λ as the ridge factor for regularization,

$$\boldsymbol{w}^* = (\mathbf{\Psi}^\top_{0:T}\mathbf{\Psi}_{0:T} + \lambda \boldsymbol{I})^{-1}\mathbf{\Psi}^\top_{0:T}\boldsymbol{Y}_{0:T}. \tag{2.7}$$

Challenges in ProMPs Despite their direct trajectory representation and probabilistic power, ProMPs suffer from a lack of intrinsic system dynamics. They do not guarantee a specified initial pose nor convergence to a desired final pose, and replanning can introduce discontinuities when parameters change.

2.1.3. Conditional Neural Movement Primitives (CNMPs)

Both DMPs and ProMPs constitute the classical movement-primitive framework for trajectory representation. In their original formulations, several strategies were proposed to adapt trajectories to new tasks or robot configurations. However, these strategies cannot cope with complex reasoning or high-dimensional inputs—such as raw images. More recent works (e.g., [33, 15]) embed these primitives within deep neural architectures, leveraging neural networks to interpret task context while preserving the smoothness and compact output dimensionality of the original MPs.

Among these next-generation methods, Conditional Neural Movement Primitives (CNMPs) [4] directly employs a Conditional Neural Processes (CNP) model [34] (shown in Fig. 2.2) as a trajectory generator to predict the trajectory distribution via an encoder–decoder network architecture. The encoder maps the observation and key task descriptors, including robot pose, desired goal, and dynamic parameters, e. g. workload—into a latent representation. Given a sequence of query times, the decoder then predicts a factorized Gaussian distribution over positions, $p(\boldsymbol{y}_{0:T}) = \prod_0^T p(y_t)$, where $p(y_t) = \mathcal{N}(\mu_t, \sigma_t^2)$. Information from multiple observations is merged through mean aggregation to produce a latent task representation. Further, this latent representation is used to condition the trajectory generation procedure.

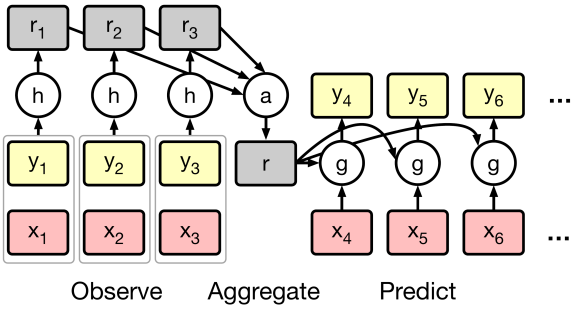


Figure 2.2.: An Illustration of Conditional Neural Processes, originally from Garnelo et al. [34]. CNMPs directly model trajectories using this architecture. In this illustration, an encoder encodes observation $x_{1:3}$ and trajectory $y_{1:3}$ into hidden states $r_{1:3}$. A mean aggregator aggregates these hidden states into a latent representation r. Then a decoder which is conditioned on the latent representation, predicts future trajectories given new observations.

Challenges in CNMPs While CNMPs enable non-linear conditioning even on high-dimensional inputs such as images, it only captures limited per-step action variance, ignoring temporal correlations within the trajectory distribution, making the sampling of diverse trajectories infeasible. Moreover, because CNMPs do not employ basis functions, the generated mean trajectory is not necessarily smooth, which can cause issues during execution.

2.1.4. Other Noticeable Movement Primitives Variants

Serving as fundamental building blocks for robot trajectory representation and generation, MPs have been widely adopted across various domains of robot learning. Table 2.1 summarizes representative studies and provides concise descriptions of the most notable approaches reported in the literature.

Table 2.1.: Noticeable Movement Primitives Related Publications in the Literature.

Method	Description
Zhou et al. [16]	Extend ProMPs for via-point enforcement in imitation learning.
Pahič et al. [22]	Use DMPs for learning digit writing trajectories from image inputs.
Bahl et al. [15]	Use DMPs for multiple step action chunks in both imitation learning and RL.
Abu-Dakka et al. [35]	Extend DMPs to Riemannian manifolds for orientation learning.
Akbulut et al. [36]	A hybrid model of CNMPs to update dataset using RL for failure case.
Rozo et al. [29]	Extend ProMPs to Riemannian manifolds for orientation learning.
Przystupa et al. [37]	Combine ProMPs and CNMPs in one model.
Daab et al. [38]	Full pose via-point behaviors in imitation learning.
Scheikl et al. [39]	Combine MPs with diffusion models in imitation learning.
Carvalho et al. [40]	Combine MPs with motion planning and diffusion models in imitation learning.
Celik et al. [41]	Use MPs for diverse skills learning in robot manipulation.
Zhang et al. [42]	Unify strike and rhythmic movements using ProMPs' framework.
Huang et al. [43]	Learn skills from demos using kernel-based representation.

2.2. Reinforcement Learning (RL)

In this section, we briefly cover the reinforcement learning fundamentals discussed in the methods developed in this dissertation, ranging from Markov Decision Processes, on- and off-policy RL, value function approximation, importance sampling, trust regions and episodic RL.

2.2.1. Markov Decision Processes (MDP)

In this dissertation, we consider a Markov Decision Process (MDP) [44] in the context of policy search, formally defined as the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{P}_0, \gamma)$. We assume the state space $\mathcal{S}$ and action space $\mathcal{A}$ are continuous. The transition probabilities $\mathcal{T} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ describe the probability of transitioning to state $\boldsymbol{s}_{t+1}$ given the current state $\boldsymbol{s}_t \in \mathcal{S}$ and action $\boldsymbol{a}_t \in \mathcal{A}$. The initial state distribution is denoted as $\mathcal{P}_0 : \mathcal{S} \to [0, 1]$. The reward $r_t(\boldsymbol{s}_t, \boldsymbol{a}_t)$ is given by the function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The discount factor $\gamma \in [0, 1]$ determines the weight of future rewards. The goal of RL is to find a policy π that maximizes the expected return

$$R = \mathbb{E}_{\mathcal{T}, \mathcal{P}_0, \pi} \left[\sum_{t=0}^{\infty} \gamma^t r_t \right]. \tag{2.8}$$

State Space. At time step t, the agent is in a state $\boldsymbol{s}_t \in \mathcal{S}$, which contains all the information required to make an optimal decision. In robotic tasks, $\boldsymbol{s}_t$ may include the robot's global pose, internal

joint angles, and positions of relevant objects in the workspace. Depending on the application, $\mathcal{S}$ can be discrete or continuous; in this dissertation, we focus on continuous state spaces, which are more common in robotics.

Action Space. The agent takes an action $\boldsymbol{a}_t \in \mathcal{A}$ at time step t to interact with the environment. The action space $\mathcal{A}$ can be discrete or continuous depending on the task. In robotics, continuous actions often represent target joint positions, velocities, or torques, which are then executed by a low-level controller. This dissertation considers continuous state-action spaces.

Dynamics. After executing $\boldsymbol{a}_t$ in $\boldsymbol{s}_t$, the environment produces the next state $\boldsymbol{s}_{t+1}$ according to the transition dynamics $\mathcal{T}$. Formally,

$$p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathcal{T}(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t), \tag{2.9}$$

which may be stochastic or deterministic. In most RL settings, $\mathcal{T}$ is unknown to the agent.

Reward. The reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ evaluates the agent's decision at each time step. We write $r_t = \mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t)$ to denote the reward after taking $\boldsymbol{a}_t$ in $\boldsymbol{s}_t$. Although in general the reward may also depend on $\boldsymbol{s}_{t+1}$, we focus on the common case $\mathcal{R}(\boldsymbol{s}_t, \boldsymbol{a}_t)$.

Initial State. At the start of each episode, the initial state $\boldsymbol{s}_0$ is sampled from the distribution $\mathcal{P}_0$. This distribution is determined by the environment and is unknown to the agent.

Markov Property. A key assumption of the MDP is the Markov property:

$$p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t, \boldsymbol{s}_{t-1}, \boldsymbol{a}_{t-1}, \ldots, \boldsymbol{s}_0) = p(\boldsymbol{s}_{t+1}|\boldsymbol{s}_t, \boldsymbol{a}_t). \tag{2.10}$$

This means that $\boldsymbol{s}_t$ contains all information needed to predict $\boldsymbol{s}_{t+1}$ given $\boldsymbol{a}_t$, without requiring the history of past states or actions. When the agent cannot directly observe the full environment state, the problem becomes a Partially Observable MDP (POMDP), where additional estimation techniques are required [45].

2.2.2. Policy Gradient

2.2.2.1. Objective.

We start by introducing a differentiable, stochastic policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)$ with parameters $\boldsymbol{\theta}$. Our goal is to maximize the expected return

$$J(\boldsymbol{\theta}) \;=\; \mathbb{E}_{\tau\sim\pi_{\boldsymbol{\theta}}}\big[R(\tau)\big], \qquad R(\tau) \;=\; \sum_{t=0}^{\infty} \gamma^t\, r_t. \tag{2.11}$$

This objective formalizes the RL problem as a standard parameter-optimization task. By taking the expectation over complete trajectories τ, we implicitly average over every random factor: the initial state distribution $\mathcal{P}_0$, the stochastic transitions $\mathcal{T}$, and the policy itself. Maximizing $J(\boldsymbol{\theta})$ therefore corresponds to finding parameters that, on average, yield high cumulative rewards.

Trajectory probability. The likelihood of a trajectory $\tau = (\boldsymbol{s}_0, \boldsymbol{a}_0, \boldsymbol{s}_1, \boldsymbol{a}_1, \dots)$ under the policy and the environment dynamics factorizes as

$$p_{\boldsymbol{\theta}}(\tau) = \mathcal{P}_0(\boldsymbol{s}_0) \prod_{t=0}^{\infty} \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)\, \mathcal{T}(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t). \tag{2.12}$$

This factorization is a direct consequence of the Markov property: each state transition depends only on the current state–action pair, and each action is drawn independently from $\pi_{\boldsymbol{\theta}}$ given the current state. Explicitly writing $p_{\boldsymbol{\theta}}(\tau)$ makes it clear which terms depend on $\boldsymbol{\theta}$ (the policy) and which do not (the environment), a distinction that becomes crucial in the gradient derivation.

Gradient of the objective. To optimize $J(\boldsymbol{\theta})$ we need its gradient:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \int p_{\boldsymbol{\theta}}(\tau)\, R(\tau)\, d\tau. \tag{2.13}$$

Interchanging $\nabla_{\boldsymbol{\theta}}$ and the integral is justified by dominated-convergence arguments because the integrand is smooth in $\boldsymbol{\theta}$ and the policy's support does not vanish abruptly. This step converts the abstract optimization problem into a concrete integral whose integrand we can manipulate.

Likelihood-ratio (score-function) trick. We rewrite the integrand gradient using

$$\nabla_{\boldsymbol{\theta}} p_{\boldsymbol{\theta}}(\tau) = p_{\boldsymbol{\theta}}(\tau)\, \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau), \tag{2.14}$$

yielding

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \int p_{\boldsymbol{\theta}}(\tau)\, R(\tau)\, \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau)\, d\tau = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \big[R(\tau)\, \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) \big]. \tag{2.15}$$

The key advantage is that the gradient now appears inside an expectation with respect to $p_{\boldsymbol{\theta}}(\tau)$ itself. Consequently, we can estimate it with Monte Carlo sampling from the current policy without requiring knowledge of $\nabla_{\boldsymbol{\theta}} \mathcal{T}$ or $\nabla_{\boldsymbol{\theta}} \mathcal{P}_0$. This property makes REINFORCE a *model-free* method.

Log-gradient of the trajectory. Since $\mathcal{P}_0$ and $\mathcal{T}$ are independent of $\boldsymbol{\theta}$, their gradients vanish:

$$\log p_{\boldsymbol{\theta}}(\tau) = \log \mathcal{P}_0(\boldsymbol{s}_0) + \sum_{t=0}^{\infty} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t) + \sum_{t=0}^{\infty} \log \mathcal{T}(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t, \boldsymbol{a}_t), \tag{2.16}$$

$$\nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau) = \sum_{t=0}^{\infty} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t). \tag{2.17}$$

Intuitively, only the action-selection mechanism depends on $\boldsymbol{\theta}$, so they are the only terms contribute to the gradient. This separation dramatically simplifies the expression.

REINFORCE estimator. Substituting back gives the REINFORCE gradient [46]:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \Big[R(\tau) \sum_{t=0}^{\infty} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t) \Big]. \tag{2.18}$$

To implement this in practice, we sample N finite-horizon trajectories of length $T^{(n)}$ and compute

$$\hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \Big(R(\tau^{(n)}) \sum_{t=0}^{T^{(n)}-1} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t^{(n)} \mid \boldsymbol{s}_t^{(n)}) \Big). \tag{2.19}$$

This estimator is *unbiased*: its expectation equals the true gradient. Its simplicity is appealing—no backpropagation through time or environment model is needed—but it typically exhibits high variance, motivating variance-reduction techniques.

2.2.2.2. Policy Gradient Theorem [47].

We split the trajectory return into rewards that precede the current action and those that follow it. For trajectory n at step t the full discounted return is

$$R(\tau^{(n)}) \;=\; \underbrace{\sum_{l=0}^{t-1} \gamma^{l}\, r_l^{(n)}}_{\text{past, independent of } \boldsymbol{a}_t^{(n)}} \;+\; \underbrace{\sum_{l=t}^{T^{(n)}-1} \gamma^{l-t}\, r_l^{(n)}}_{\text{future, influenced by } \boldsymbol{a}_t^{(n)}}. \tag{2.20}$$

We define the discounted future return, i. e. *reward to come*

$$G_t^{(n)} \;=\; \sum_{l=t}^{T^{(n)}-1} \gamma^{l-t}\, r_l^{(n)}, \tag{2.21}$$

which contains only the rewards that the current action $\boldsymbol{a}_t^{(n)}$ can still affect. Because the multiplicative factor γ^{l-t} matches the discounting in the objective $J(\boldsymbol{\theta})$, $G_t^{(n)}$ is an unbiased estimator of the action–value $Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t^{(n)}, \boldsymbol{a}_t^{(n)})$, i. e. estimation of the expected reward to come.

Using the same full return $R(\tau^{(n)})$ for every time step indeed gives an *unbiased* gradient, but its variance is larger than that obtained with $G_t^{(n)}$. To see why, compare the two per–time-step gradient contributions

$$g_t \;=\; R(\tau^{(n)})\,\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t^{(n)} \mid \boldsymbol{s}_t^{(n)}), \qquad \tilde{g}_t \;=\; G_t^{(n)}\,\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t^{(n)} \mid \boldsymbol{s}_t^{(n)}). \tag{2.22}$$

Because g_t multiplies the score term by past rewards that are (i) uninfluenced by $\boldsymbol{a}_t^{(n)}$ and (ii) often weakly correlated with it, those extra terms act like additional noise. Formally[3],

$$\mathrm{Var}[\tilde{g}_t] \;=\; \mathrm{Var}[g_t] \;-\; 2\sum_{l<t} \mathrm{Cov}\Big(r_l^{(n)},\; \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\big(\boldsymbol{a}_t^{(n)} \mid \boldsymbol{s}_t^{(n)}\big)\Big) \;-\; \mathrm{Var}\Big(\sum_{l<t} \gamma^{\,l}\, r_l^{(n)}\Big), \tag{2.23}$$

so $\mathrm{Var}[\tilde{g}_t] < \mathrm{Var}[g_t]$ because the omitted past rewards typically have zero covariance with the gradient in stochastic environments.

Hence, replacing $R(\tau^{(n)})$ with the step-specific $G_t^{(n)}$ preserves unbiasedness while **reducing** variance, yielding a more efficient policy-gradient estimator and laying the foundation for further variance-reduction techniques such as learned baselines and actor–critic methods.

2.2.2.3. Variance reduction with baselines [47].

Another common trick is to subtract a baseline $b(\boldsymbol{s}_t)$ that does not depend on $\boldsymbol{a}_t$:

$$\hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T^{(n)}-1} \big(G_t^{(n)} - b(\boldsymbol{s}_t^{(n)})\big)\, \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t^{(n)} \mid \boldsymbol{s}_t^{(n)}), \tag{2.24}$$

where $G_t^{(n)} = \sum_{l=t}^{T^{(n)}-1} \gamma^{l-t} r_l^{(n)}$ is the empirical return from time t, i. e. reward to come. Because the baseline is independent of the action, the expectation of the gradient remains unchanged, but the variance can be significantly reduced [48]. A proof can be conducted as

$$\begin{aligned} \mathbb{E}_{\tau\sim p_\theta(\tau)}\big[b(\boldsymbol{s}_t)\,\nabla_\theta \log p_\theta(\tau)\big] &= \int p_\theta(\tau)\, b(\boldsymbol{s}_t)\,\nabla_\theta \log p_\theta(\tau)\, d\tau = \int b(\boldsymbol{s}_t)\,\nabla_\theta p_\theta(\tau)\, d\tau \\ &= b(\boldsymbol{s}_t)\,\nabla_\theta \int p_\theta(\tau)\, d\tau = b(\boldsymbol{s}_t)\,\nabla_\theta 1 = 0. \end{aligned} \tag{2.25}$$

[3] Here, we utilize the property $\mathrm{Var}(X+Y) \;=\; \mathrm{Var}(X) \;+\; \mathrm{Var}(Y) \;+\; 2\;\mathrm{Cov}(X,Y)$.

Choosing $b(\boldsymbol{s}_t)$ as an estimate of the state value function $V^\pi(s_t) = \mathbb{E}\Big[\sum_{l=0}^{\infty} r_{t+l}\Big]$ leads to the well-known actor–critic family of algorithms, which further improves sample efficiency.

$$\hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T^{(n)}-1} \big(G_t^{(n)} - V^\pi(s_t)\big) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t^{(n)} \mid \boldsymbol{s}_t^{(n)}), \tag{2.26}$$

2.2.2.4. Use Q-values for Policy Gradient

Observing that $G_t^{(n)} = \sum_{l=t}^{T^{(n)}-1} \gamma^{l-t} r_l^{(n)}$ is an unbiased estimator of the $Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t^{(n)}, \boldsymbol{a}_t^{(n)})$, i. e. estimation of the expected reward to come, we have

$$\hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T^{(n)}-1} \big(Q^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t^{(n)}, \boldsymbol{a}_t^{(n)}) - V^\pi(s_t^{(n)})\big) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t^{(n)} \mid \boldsymbol{s}_t^{(n)}), \tag{2.27}$$

$$\hat{\nabla}_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=0}^{T^{(n)}-1} A^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_t^{(n)}, \boldsymbol{a}_t^{(n)}) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t^{(n)} \mid \boldsymbol{s}_t^{(n)}). \tag{2.28}$$

Here, we use the definition of the advantage function $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$, which centers the learning signal around zero for further variance reduction.

2.2.2.5. Summary of Policy-Gradient Formulation.

Schulman et al. [49] summarized the policy-gradient objective as optimizing the expected discounted return by estimating

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \nabla_{\boldsymbol{\theta}} \mathbb{E}\Big[\sum_{t=0}^{\infty} r_t\Big] = \mathbb{E}\Big[\sum_{t=0}^{\infty} \Omega_t \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)\Big], \tag{2.29}$$

where the *return signal* Ω_t can be chosen in several equivalent ways to trade bias for variance [48] and enable different algorithmic families. Common choices for Ω_t are:

1. $\sum_{l=0}^{\infty} r_l$: total trajectory reward (high variance, unbiased).
2. $\sum_{l=t}^{\infty} r_l$: future reward following $\boldsymbol{a}_t$ (lower variance than choice 1).
3. $\sum_{l=t}^{\infty} r_l - b(\boldsymbol{s}_t)$: state-dependent baseline subtraction.
4. $Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) = \mathbb{E}\Big[\sum_{l=0}^{\infty} r_{t+l}\Big]$: state–action value function.

5. $A^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) = Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t) - V^\pi(\boldsymbol{s}_t)$: advantage function.
6. $r_t + V^\pi(\boldsymbol{s}_{t+1}) - V^\pi(\boldsymbol{s}_t)$: one-step temporal-difference (TD) residual; introduces bootstrapping for lower variance at the cost of bias that vanishes as the value estimator improves.

Equation (2.29) shows that any Ω_t whose expectation equals the true action-value $Q^\pi(\boldsymbol{s}_t, \boldsymbol{a}_t)$ yields an unbiased gradient. Different choices mainly affect estimator variance and thus sample efficiency, leading to a spectrum of algorithms ranging from Monte-Carlo REINFORCE (Choices 1–3) to actor–critic and TD methods (Choices 4–6).

2.2.3. Policy Updates within Trust Regions

2.2.3.1. Fundamentals of Importance Sampling.

Importance Sampling (IS) [50] is a Monte-Carlo technique that lets us estimate an expectation with respect to a *target* distribution $p(\boldsymbol{x})$ while drawing samples from a different, easier-to-sample *proposal* distribution $q(\boldsymbol{x})$. Assume $p(\boldsymbol{x}) > 0$ whenever $q(\boldsymbol{x}) > 0$ (the *support* condition). For any integrable function $f(\boldsymbol{x})$ we have

$$\mathbb{E}_{\boldsymbol{x}\sim p}[f(\boldsymbol{x})] \;=\; \int f(\boldsymbol{x})\, p(\boldsymbol{x})\, d\boldsymbol{x} \;=\; \int f(\boldsymbol{x})\, \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\, q(\boldsymbol{x})\, d\boldsymbol{x} \;=\; \mathbb{E}_{\boldsymbol{x}\sim q}\big[\rho(\boldsymbol{x})\, f(\boldsymbol{x})\big], \tag{2.30}$$

where the importance weight

$$\rho(\boldsymbol{x}) \;=\; \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \tag{2.31}$$

corrects for the mismatch between p and q. Given M samples $\{\boldsymbol{x}^{(i)}\}_{i=1}^{M}$ from q, the Monte-Carlo estimator

$$\hat{\mu}_{\text{IS}} \;=\; \frac{1}{M} \sum_{i=1}^{M} \rho(\boldsymbol{x}^{(i)})\, f(\boldsymbol{x}^{(i)}) \tag{2.32}$$

is unbiased: $\mathbb{E}_q[\hat{\mu}_{\text{IS}}] = \mathbb{E}_p[f(\boldsymbol{x})]$. However, the variance of $\hat{\mu}_{\text{IS}}$ depends on how strongly the weights vary:

$$\operatorname{Var}_q\big[\hat{\mu}_{\text{IS}}\big] \;=\; \frac{1}{M} \Big(\mathbb{E}_q\big[\rho^2 f^2\big] - \mathbb{E}_p[f]^2\Big). \tag{2.33}$$

If p and q differ too much, some $\rho(\boldsymbol{x})$ become very large, inflating the variance and making the estimator unreliable. An example is shown in Fig. 2.3. A common rule of thumb is to choose q as close as possible to p in regions where $f(\boldsymbol{x})$ is large.

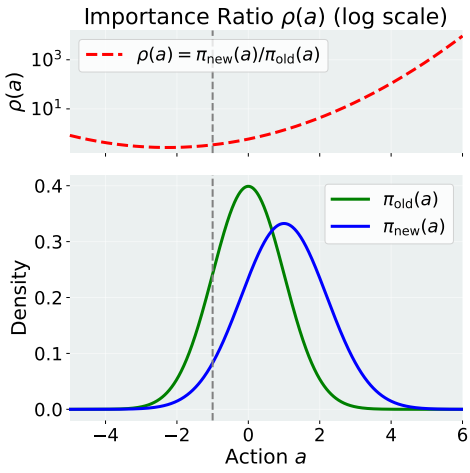


Figure 2.3.: Importance sampling ratio of two Gaussian distributions, $\mu_1 = 0, \sigma_1 = 1$ and $\mu_2 = 1, \sigma_2 = 1.2$ respectively.

2.2.3.2. Importance-Sampling Policy Gradient

To improve sample efficiency, we reuse trajectories collected by an *old* policy π_{old} to update a *new* policy $\pi_{\boldsymbol{\theta}}$. To keep the gradient estimator unbiased under this slight off-policy setting[4] we weight each sample by an importance ratio. For a full trajectory τ the likelihood ratio is

$$\rho(\tau) \;=\; \frac{p_{\boldsymbol{\theta}}(\tau)}{p_{\text{old}}(\tau)} = \prod_{t=0}^{T-1} \frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)}{\pi_{\text{old}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)}. \tag{2.34}$$

Substituting $\rho(\tau)$ into the REINFORCE identity gives

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim p_{\text{old}}}\Big[\rho(\tau)\, R(\tau)\, \nabla_{\boldsymbol{\theta}} \log p_{\boldsymbol{\theta}}(\tau)\Big]. \tag{2.35}$$

Using the decomposition $\log p_{\boldsymbol{\theta}}(\tau) = \sum_t \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)$ and pulling terms that *contain* $\boldsymbol{\theta}$ inside the expectation yields the per-step form

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim p_{\text{old}}}\Big[\sum_{t=0}^{T-1} \rho_t\, G_t\, \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)\Big], \qquad \rho_t = \frac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)}{\pi_{\text{old}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)}. \tag{2.36}$$

Here $G_t = \sum_{l=t}^{T-1} \gamma^{\,l-t} r_l$ is the reward-to-come. Both estimators remain *unbiased*, yet the products of ratios can grow exponentially with T; large ρ_t inflate Monte-Carlo variance and slow learning.

[4] Within RL parlance, such methods are still considered "on-policy" because the data come from the most recent policy, not from a long-term replay buffer.

Several practical stabilizers enforce safe policy updates:

Natural gradient [51]. Kakade [52] propose the step $F^{-1}\nabla_{\boldsymbol{\theta}} J$ with the Fisher information matrix [53]

$$F = \mathbb{E}_{\boldsymbol{s},\boldsymbol{a}}\big[\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s})\, \nabla_{\boldsymbol{\theta}}^{\top} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s})\big], \tag{2.37}$$

which is equivalent to solving a first-order KL-constrained [54] problem.

KL trust region. Schulman et al. [55] restrict the update so that

$$\mathbb{E}_{\boldsymbol{s}\sim d^{\pi_{\text{old}}}}\big[D_{\mathrm{KL}}\big(\pi_{\text{old}}(\cdot \mid \boldsymbol{s}) \,\|\, \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s})\big)\big] \le \delta, \tag{2.38}$$

indirectly bounding ρ_t because D_{KL} upper-bounds the expected log-ratio.

Ratio clipping. Schulman et al. [7] replace ρ_t in Eq. (2.36) with $\mathrm{clip}\big(\rho_t,\, 1-\epsilon,\, 1+\epsilon\big)$ to cap outliers and obtain PPO.

2.2.3.3. Trust-Region Policy Optimization (TRPO) [55].

TRPO frames the update as a constrained optimization problem that maximizes a first-order surrogate while bounding the average Kullback–Leibler (KL) divergence from the behavior policy:

$$\max_{\boldsymbol{\theta}} \; \hat{\mathbb{E}}\Big[\rho_t\, A_t^{\pi_{\text{old}}}\Big] \quad \text{s.t.} \quad \hat{\mathbb{E}}\Big[D_{\mathrm{KL}}\big(\pi_{\text{old}} \,\|\, \pi_{\boldsymbol{\theta}}\big)\Big] \le \delta. \tag{2.39}$$

Here $A_t^{\pi_{\text{old}}} = G_t - V^{\pi_{\text{old}}}(\boldsymbol{s}_t)$ is the empirical advantage under the old policy. The hard KL constraint defines a *trust region* that keeps ρ_t close to one, preventing variance blow-up.

Solving Eq. (2.39). Linearizing the objective and taking a second-order Taylor expansion of the constraint give the quadratic program

$$\max_{\Delta\boldsymbol{\theta}} \; g^{\top}\Delta\boldsymbol{\theta} \quad \text{s.t.} \quad \Delta\boldsymbol{\theta}^{\top} F\, \Delta\boldsymbol{\theta} \le \delta, \tag{2.40}$$

where $g = \hat{\mathbb{E}}[\rho_t\, A_t^{\pi_{\text{old}}} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}]$ and F is the empirical Fisher matrix. Eq (2.40) yields the natural-gradient step $\Delta\boldsymbol{\theta}^{*} = \sqrt{\delta / (g^{\top} F^{-1} g)}\, F^{-1} g$. TRPO computes $F^{-1}g$ with conjugate gradients [56], then performs a back-tracking line search [57] to enforce the δ bound and guarantee monotonic policy improvement.

Algorithm 1 Proximal Policy Optimization (PPO)

Init: policy $\pi_{\boldsymbol{\theta}}$, value function $V_{\boldsymbol{\phi}}$, clip parameter ϵ, learning rates α_π, α_V
repeat
  Collect trajectories $\{\boldsymbol{s}_t, \boldsymbol{a}_t, r_t\}_{t=0}^{T}$ by running $\pi_{\boldsymbol{\theta}}$
  Compute advantages $\hat{A}_t$ (e.g., via GAE [49]) and returns $\hat{R}_t$
  for K epochs **do**
    Sample minibatch $\{\boldsymbol{s}_i, \boldsymbol{a}_i, \hat{A}_i, \hat{R}_i\}$
    $\rho_i \leftarrow \dfrac{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_i \mid \boldsymbol{s}_i)}{\pi_{\text{old}}(\boldsymbol{a}_i \mid \boldsymbol{s}_i)}$
    Policy loss: $L^{\text{CLIP}}(\boldsymbol{\theta}) = \dfrac{1}{B}\sum_i \min\Big(\rho_i \hat{A}_i,\ \text{clip}(\rho_i, 1-\epsilon, 1+\epsilon)\,\hat{A}_i\Big)$
    Value loss: $L^{V}(\boldsymbol{\phi}) = \dfrac{1}{B}\sum_i \big(V_{\boldsymbol{\phi}}(\boldsymbol{s}_i) - \hat{R}_i\big)^2$
    Update $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_\pi \nabla_{\boldsymbol{\theta}} L^{\text{CLIP}}(\boldsymbol{\theta})$
    Update $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \alpha_V \nabla_{\boldsymbol{\phi}} L^{V}(\boldsymbol{\phi})$
  end for
  $\pi_{\text{old}} \leftarrow \pi_{\boldsymbol{\theta}}$
until converged

2.2.3.4. Proximal Policy Optimization (PPO) [7].

PPO retains the trust-region idea but replaces the hard constraint with a surrogate loss that is differentiable and amenable to stochastic gradient descent:

$$L^{\text{CLIP}}(\boldsymbol{\theta}) = \hat{\mathbb{E}}\Big[\min\big(\rho_t\, A_t^{\pi_{\text{old}}},\ \text{clip}\big(\rho_t,\ 1-\epsilon,\ 1+\epsilon\big)\, A_t^{\pi_{\text{old}}}\big)\Big]. \tag{2.41}$$

The clipping operation limits ρ_t to the interval $[1-\epsilon,\ 1+\epsilon]$, preventing excessively large or small importance ratios and thus implicitly enforcing a soft trust region. A second variant adds a KL penalty:

$$L^{\text{KL-PEN}}(\boldsymbol{\theta}) = \hat{\mathbb{E}}\Big[\rho_t\, A_t^{\pi_{\text{old}}} - \beta\, D_{\text{KL}}\big(\pi_{\text{old}} \,\|\, \pi_{\boldsymbol{\theta}}\big)\Big], \tag{2.42}$$

where β is an adaptive penalty coefficient. PPO alternates several epochs of minibatch SGD [58] on either Eq. (2.41) or Eq. (2.42), using the same batch of trajectories, and typically attains performance comparable to TRPO while being simpler to implement and faster in wall-clock time. An algorithm box of PPO is presented in Algorithm 1.

Connections and take-aways. Natural Policy Gradient, TRPO, and PPO all emerge from controlling the variance of importance ratios ρ_t by restricting the policy move in KL space. TRPO does so via an explicit constraint solved with natural-gradient tools, whereas PPO enforces a softer, first-order approximation through clipping or a penalty term. Both methods preserve the unbiasedness of the importance-sampling gradient while achieving stable and efficient learning in high-dimensional control problems.

2.2.4. Differentiable Trust-Region Layers

2.2.4.1. Motivation

Unlike TRPO and PPO, which constrain only the *expected* KL divergence between successive policies, the differentiable Trust-Region Projection Layer (TRPL) of Otto et al. [59] enforces the bound *per state*. TRPL inserts a closed-form projection operator at the end of a Gaussian policy network, mapping each provisional mean–covariance update back into a predefined divergence domain. This exact, state-wise correction eliminates line searches, conjugate-gradient solves, and heuristic ratio clipping, yielding a lighter implementation and faster training. Because every forward pass already satisfies the trust radius, TRPL allows larger steps without sacrificing stability and achieves stronger returns with minimal hyper-parameter tuning on standard control benchmarks.

Let the old policy be $\pi_{\text{old}}(\boldsymbol{a} \mid \boldsymbol{s}) = \mathcal{N}\big(\boldsymbol{\mu}_{\text{old}}(\boldsymbol{s}),\, \boldsymbol{\Sigma}_{\text{old}}(\boldsymbol{s})\big)$ and the provisional update be $\pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s}) = \mathcal{N}\big(\boldsymbol{\mu}(\boldsymbol{s}),\, \boldsymbol{\Sigma}(\boldsymbol{s})\big)$. TRPL projects the Gaussian parameters $\boldsymbol{\mu}, \boldsymbol{\Sigma}$ to $\tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\Sigma}}$, such that

$$\tilde{\pi}^* = \arg\min_{\tilde{\pi}} D\big(\tilde{\pi} \,\|\, \pi_{\boldsymbol{\theta}}\big) \quad \text{s.t.} \quad D\big(\tilde{\pi} \,\|\, \pi_{\text{old}}\big) \le \epsilon, \quad H(\tilde{\pi}) \ge \beta, \tag{2.43}$$

where $D(\cdot\|\cdot)$ may be KL [54], Wasserstein-2 [60], or Frobenius distance[61] and $H(\cdot)$ is the entropy lower-bounded by β. Following [62, 63], the constrained problem admits a closed-form solution in natural parameters with multipliers η, ω. In natural form $\boldsymbol{q} = \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}$ and $\boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1}$; for the KL case the projection is

$$\tilde{\boldsymbol{\Lambda}} = \frac{\eta\, \boldsymbol{\Lambda}_{\text{old}} + \boldsymbol{\Lambda}}{\eta + 1 + \omega}, \qquad \tilde{\boldsymbol{q}} = \frac{\eta\, \boldsymbol{q}_{\text{old}} + \boldsymbol{q}}{\eta + 1 + \omega}, \qquad \tilde{\boldsymbol{\Sigma}} = \tilde{\boldsymbol{\Lambda}}^{-1}, \quad \tilde{\boldsymbol{\mu}} = \tilde{\boldsymbol{\Sigma}}\, \tilde{\boldsymbol{q}}. \tag{2.44}$$

In practice, TRPL operates on the Gaussian policy outputs—the mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$—and enforces trust regions through a state-specific projection. Let the old policy be $\pi_{\text{old}}(\boldsymbol{a} \mid \boldsymbol{s}) = \mathcal{N}\big(\boldsymbol{\mu}_{\text{old}}(\boldsymbol{s}),\, \boldsymbol{\Sigma}_{\text{old}}(\boldsymbol{s})\big)$ and the provisional update be $\pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s}) = \mathcal{N}\big(\boldsymbol{\mu}(\boldsymbol{s}), \boldsymbol{\Sigma}(\boldsymbol{s})\big)$. The adjusted parameters $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\Sigma}}$ satisfy the per-state optimization

$$\begin{aligned} &\arg\min_{\tilde{\boldsymbol{\mu}}_{\boldsymbol{s}}} \; d_{\text{mean}}\big(\tilde{\boldsymbol{\mu}}_{\boldsymbol{s}},\, \boldsymbol{\mu}(\boldsymbol{s})\big) \quad \text{s.t.} \quad d_{\text{mean}}\big(\tilde{\boldsymbol{\mu}}_{\boldsymbol{s}},\, \boldsymbol{\mu}_{\text{old}}(\boldsymbol{s})\big) \le \epsilon_{\mu}, \\ &\arg\min_{\tilde{\boldsymbol{\Sigma}}_{\boldsymbol{s}}} \; d_{\text{cov}}\big(\tilde{\boldsymbol{\Sigma}}_{\boldsymbol{s}},\, \boldsymbol{\Sigma}(\boldsymbol{s})\big) \quad \text{s.t.} \quad d_{\text{cov}}\big(\tilde{\boldsymbol{\Sigma}}_{\boldsymbol{s}},\, \boldsymbol{\Sigma}_{\text{old}}(\boldsymbol{s})\big) \le \epsilon_{\Sigma}. \end{aligned} \tag{2.45}$$

If the unconstrained parameters $\boldsymbol{\mu}(\boldsymbol{s})$ or $\boldsymbol{\Sigma}(\boldsymbol{s})$ violate the bounds $\epsilon_{\mu}, \epsilon_{\Sigma}$, TRPL projects them back onto the trust-region boundary, ensuring stable updates. A TRPL projection pipeline originally reported in [59] can be found in Fig 2.4.

2.2.4.2. Algorithmic Use

The projection is wrapped as the *last* layer of the policy network and is fully differentiable; gradients flow through the closed-form map (or its implicit gradients for KL projection). Training therefore

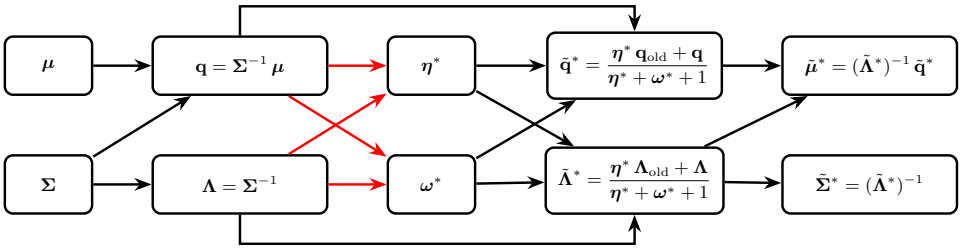


Figure 2.4.: Compute graph of the KL projection layer, originally reported by Otto et al. [59]. Starting from the provisional policy's mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$, the layer first converts them into their natural parameters. It then solves the dual problem to obtain the optimal Lagrange multipliers (η, ω), which yield the projected natural parameters, an analytic interpolation between the old and provisional policies. $\longrightarrow$ denotes closed-form computations; $\rightarrow$ denotes the numerical optimization of the dual.

proceeds with standard SGD / Adam [58, 64] while guaranteeing that every forward pass respects the state-wise trust region. Because it is a pure projection layer, TRPL can complement any policy-gradient or actor–critic method that outputs Gaussian means and covariances.

Advantages over TRPO / PPO

- **Exact constraint.** No linearization or clipping—projection enforces the radius η by construction.
- **State-wise control.** Bounds the worst-case change $\max_{\boldsymbol{s}} D\big(\pi_{\text{old}}(\cdot \mid \boldsymbol{s}) \,\|\, \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s})\big)$, whereas TRPO/PPO regulate only an expectation.
- **Implementation simplicity.** A single layer replaces conjugate-gradient (TRPO) or hand-tuned clipping coefficients (PPO), and is less sensitive to implementation details [65].
- **Flexibility.** Frobenius and 2-Wasserstein variants avoid the root-finding needed for KL projection and can encourage higher entropy for exploration.

2.2.5. Off-Policy Reinforcement Learning

On- vs. Off-Policy. In *on-policy* methods the data-generating policy and the *current policy* being optimized coincide; updates rely solely on freshly collected trajectories. Off-policy methods [66] decouple these roles: a *behavior policy* $\pi_b(\boldsymbol{a} \mid \boldsymbol{s})$ gathers experience, while a distinct *current policy* $\pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s})$ is improved using that experience [67, 45]. Key motivations for off-policy RL include:

- *Sample Efficiency.* Replaying past transitions—often millions stored in a buffer—dramatically reduces required real interactions [68].
- *Flexible Exploration.* π_b can remain highly stochastic (e.g., ε-greedy [66]), while $\pi_{\boldsymbol{\theta}}$ converges to a deterministic or risk-aware solution.
- *Learning from Logged Data.* Offline (batch) RL [67, 69] leverages fixed datasets collected by other agents or humans—impossible for strict on-policy algorithms.

- *Stability with Function Approximation [70, 71].* Experience replay decorrelates samples, producing i.i.d.-like mini-batches that suit gradient optimizers.

The core challenge is *distribution mismatch*: transitions $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1}, d)$ come from π_b, not $\pi_{\boldsymbol{\theta}}$, so naive updates are biased. Here, d denotes a termination flag. Off-policy TD [72] and policy-gradient methods introduce *correction mechanisms*—importance-sampling ratios, target networks [73, 68], or conservative objectives [74]—to manage the resulting bias–variance trade-off.

Experience Replay Buffer. First proposed by Lin [67], Lin [75], an experience replay buffer $\mathcal{B}$ stores transitions from π_b. Mini-batches sampled uniformly or with priority [76] update the current policy $\pi_{\boldsymbol{\theta}}$. Replay breaks temporal correlations, lowers gradient variance, and reuses data many times, boosting sample efficiency [68]. Works like [77] also proposed to relabel unsuccessful experience in the replay buffer using their actual outcomes.

Target Networks. The combination of bootstrapping, off-policy data, and nonlinear critics can diverge—the "deadly triad." Deep Q-Network (DQN) stabilizes learning with a slowly updated *target critic* whose parameters are denoted $\boldsymbol{\phi}_{\text{tar}}$. The online critic uses parameters $\boldsymbol{\phi}$, and every K steps the target is set to $\boldsymbol{\phi}_{\text{tar}} \leftarrow \boldsymbol{\phi}$ [68]. Together, experience replay $\mathcal{B}$ and target parameters $\boldsymbol{\phi}_{\text{tar}}$ are fundamental safeguards for off-policy TD learning.

Q-Learning Fundamentals. In the tabular case, *Q-learning* performs the stochastic update

$$Q_{k+1}(\boldsymbol{s}_t, \boldsymbol{a}_t) \;=\; Q_k(\boldsymbol{s}_t, \boldsymbol{a}_t) + \alpha\big[r_t + \gamma \max_{\boldsymbol{a}'} Q_k(\boldsymbol{s}_{t+1}, \boldsymbol{a}') - Q_k(\boldsymbol{s}_t, \boldsymbol{a}_t)\big] \tag{2.46}$$

toward the off-policy Bellman optimum [78]; with a small step size α and sufficient exploration, it converges. When the value function is approximated by a neural critic $Q_{\boldsymbol{\phi}}$, the gradient update

$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \alpha \nabla_{\boldsymbol{\phi}} \Big(r_t + \gamma \max_{\boldsymbol{a}'} Q_{\boldsymbol{\phi}}(\boldsymbol{s}_{t+1}, \boldsymbol{a}') - Q_{\boldsymbol{\phi}}(\boldsymbol{s}_t, \boldsymbol{a}_t) \Big)^2 \tag{2.47}$$

introduces the *deadly triad*—bootstrapping, off-policy data, and nonlinear function approximation—which can cause divergence [45]. Moreover, the internal $\max$ reuses identical estimates for action selection and evaluation, yielding systematic *over-estimation* [79, 80]. Deep Q-Network (DQN) [68] mitigates instability by computing its target with a slowly updated *target critic* $Q_{\boldsymbol{\phi}_{\text{tar}}}$:

$$y_t \;=\; r_t + \gamma \max_{\boldsymbol{a}'} Q_{\boldsymbol{\phi}_{\text{tar}}}(\boldsymbol{s}_{t+1}, \boldsymbol{a}'), \tag{2.48}$$

and copying $\boldsymbol{\phi}_{\text{tar}} \leftarrow \boldsymbol{\phi}$ every K steps. Even so, over-estimation remains a core issue, motivating refinements such as Double Q-Learning [81].

Double Q-Learning. *Q-learning* over-estimates action values because the same sample is used to *select* the maximising action and to *evaluate* its value, i.e. $\mathbb{E}\big[\max_{\boldsymbol{a}} Q(\boldsymbol{s}, \boldsymbol{a})\big] \geq \max_{\boldsymbol{a}} \mathbb{E}[Q(\boldsymbol{s}, \boldsymbol{a})]$.

Double Q-learning [81] removes this bias by maintaining two independent critics, Q_{ϕ^A} and Q_{ϕ^B}, updated in alternation. When critic A is updated, critic B supplies the bootstrap target—and *vice-versa*:

$$y_t^A = r_t + \gamma\, Q_{\phi^B}\Big(\boldsymbol{s}_{t+1}, \arg\max_{\boldsymbol{a}'} Q_{\phi^A}(\boldsymbol{s}_{t+1}, \boldsymbol{a}')\Big), \tag{2.49}$$

$$\phi^A \leftarrow \phi^A - \alpha \nabla_{\phi^A}\big(y_t^A - Q_{\phi^A}(\boldsymbol{s}_t, \boldsymbol{a}_t)\big)^2. \tag{2.50}$$

Deep Double DQN [82] applies the same idea with an online critic Q_ϕ and a slowly updated target critic $Q_{\phi_{\text{tar}}}$, yielding

$$y_t = r_t + \gamma\, Q_{\phi_{\text{tar}}}\Big(\boldsymbol{s}_{t+1}, \arg\max_{\boldsymbol{a}'} Q_{\phi}(\boldsymbol{s}_{t+1}, \boldsymbol{a}')\Big). \tag{2.51}$$

Decoupling action selection from evaluation eliminates the positive bias, producing an *unbiased* and more stable estimator. Empirically, Double Q-learning reduces over-estimation errors and improves performance on Atari and continuous-control benchmarks, while adding negligible computational cost; it integrates smoothly with distributional critics [83], prioritized replay, and offline RL settings [74, 84].

Multi-Step Off-Policy Targets One-step TD targets bootstrap aggressively but discard all information beyond the next step, while Monte-Carlo targets capture the full return yet suffer high variance. *Multi-step* methods strike a balance by mixing N observed rewards with a bootstrap term. For an *on-policy* algorithm—where data are generated by the very policy being improved—the N-step return bootstraps after N observed rewards without any importance weighting:

$$\text{On-policy:} \qquad G_t^{(N),\text{on}} = \sum_{n=0}^{N-1} \gamma^n\, r_{t+n} \;+\; \gamma^N\, V_\phi\big(\boldsymbol{s}_{t+N}\big). \tag{2.52}$$

In the off-policy setting the rewards were generated by the behaviour policy π_b, so each step is reweighted by an importance ratio $\rho_t = \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_t \mid \boldsymbol{s}_t)/\pi_b(\boldsymbol{a}_t \mid \boldsymbol{s}_t)$. yielding the *raw* off-policy target [85]

$$\text{Off-policy:} \qquad G_t^{(N),\text{off}} = \sum_{n=0}^{N-1} \Big(\prod_{j=0}^{n} \rho_{t+j}\Big)\gamma^n\, r_{t+n} \;+\; \Big(\prod_{j=0}^{N-1} \rho_{t+j}\Big)\gamma^N\, V_\phi\big(\boldsymbol{s}_{t+N}\big). \tag{2.53}$$

Equation (2.53) is unbiased under off-policy settings yet its product of ratios can explode, motivating clipped or truncated variants such as Retrace[86] and V-Trace[87] to keep bias–variance in check.

Algorithm 2 Soft Actor–Critic (SAC)

Init: actor $\pi_{\boldsymbol{\theta}}$, twin critics $Q_{\boldsymbol{\phi}^1}, Q_{\boldsymbol{\phi}^2}$, targets $Q_{\boldsymbol{\phi}^1_{\text{tar}}}, Q_{\boldsymbol{\phi}^2_{\text{tar}}}$, temperature α, replay buffer $\mathcal{B}$
for each environment step **do**
  Sample $\boldsymbol{a}_t \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}_t)$, observe r_t, $\boldsymbol{s}_{t+1}$, store $(\boldsymbol{s}_t, \boldsymbol{a}_t, r_t, \boldsymbol{s}_{t+1})$ in $\mathcal{B}$
end for
for each update step **do**
  Sample batch $\{(\boldsymbol{s}_i, \boldsymbol{a}_i, r_i, \boldsymbol{s}'_i)\} \sim \mathcal{B}$
  $\boldsymbol{a}'_i \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}'_i)$
  $y_i \leftarrow r_i + \gamma\big(\min_j Q_{\boldsymbol{\phi}^j_{\text{tar}}}(\boldsymbol{s}'_i, \boldsymbol{a}'_i) - \alpha \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}'_i \mid \boldsymbol{s}'_i)\big)$
  Update $\boldsymbol{\phi}^j$ to minimize $\frac{1}{B}\sum_i (Q_{\boldsymbol{\phi}^j}(\boldsymbol{s}_i, \boldsymbol{a}_i) - y_i)^2$ for $j = 1, 2$
  $\boldsymbol{a}_i \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}_i)$ (via reparameterisation)
  Update $\boldsymbol{\theta}$ to minimize $\frac{1}{B}\sum_i \big(\alpha \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_i \mid \boldsymbol{s}_i) - \min_j Q_{\boldsymbol{\phi}^j}(\boldsymbol{s}_i, \boldsymbol{a}_i)\big)$
  (Optional) Update α using gradient of $-\alpha(\log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_i \mid \boldsymbol{s}_i) + \mathcal{H}_{\text{tar}})$
  Update targets: $\boldsymbol{\phi}^j_{\text{tar}} \leftarrow \tau \boldsymbol{\phi}^j + (1-\tau)\boldsymbol{\phi}^j_{\text{tar}}$ for $j = 1, 2$
end for

2.2.6. Noticeable Off-Policy RL Methods for Continuous Control

Deep Deterministic Policy Gradient (DDPG) [88]. DDPG maintains an *actor* $\pi_{\boldsymbol{\theta}} : \boldsymbol{s} \mapsto \boldsymbol{a}$ and a *critic* $Q_{\boldsymbol{\phi}} : (\boldsymbol{s}, \boldsymbol{a}) \mapsto \mathbb{R}$, together with their slowly drifting targets $(\pi_{\boldsymbol{\theta}_{\text{tar}}}, Q_{\boldsymbol{\phi}_{\text{tar}}})$. During data collection the behaviour policy adds Ornstein–Uhlenbeck noise, $\tilde{\boldsymbol{a}}_t = \pi_{\boldsymbol{\theta}}(\boldsymbol{s}_t) + \boldsymbol{\epsilon}_t,\ \boldsymbol{\epsilon}_t \sim \mathrm{OU}(0, \sigma)$, or per-episode parameter noise; noise is disabled at evaluation time.

Twin Delayed DDPG (TD3) [89]. TD3 stabilises DDPG via three key changes: *(i) Twin critics* $Q_{\boldsymbol{\phi}^1}, Q_{\boldsymbol{\phi}^2}$ with the *min* used in the target; *(ii) Target-policy smoothing*, adding clipped Gaussian noise $\boldsymbol{\epsilon} \sim \mathrm{clip}\big(\mathcal{N}(0, \sigma^2), -c, c\big)$ to the target action; *(iii) Delayed actor updates*, updating $\pi_{\boldsymbol{\theta}}$ and the targets every d critic steps. During data collection TD3 still applies simple Gaussian exploration noise to the actor's output.

Soft Actor–Critic (SAC) [8]. SAC employs a stochastic policy and maximises the entropy-augmented objective $J = \sum_t \mathbb{E}\big[r_t + \alpha\, \mathcal{H}\big(\pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}_t)\big)\big]$, making exploration intrinsic. Twin critics are trained with

$$y_t = r_t + \gamma \left(\min_{i=1,2} Q_{\boldsymbol{\phi}^i_{\text{tar}}}(\boldsymbol{s}_{t+1}, \boldsymbol{a}') - \alpha \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}' \mid \boldsymbol{s}_{t+1}) \right), \quad \boldsymbol{a}' \sim \pi_{\boldsymbol{\theta}}(\cdot \mid \boldsymbol{s}_{t+1}). \tag{2.54}$$

The actor uses the reparameterisation $\boldsymbol{a} = \tanh\big(f_{\boldsymbol{\theta}}(\boldsymbol{s}, \boldsymbol{\epsilon})\big)$ and minimises $\mathbb{E}\big[\alpha \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a} \mid \boldsymbol{s}) - \min_i Q_{\boldsymbol{\phi}^i}(\boldsymbol{s}, \boldsymbol{a})\big]$. An automatic entropy tuner updates α to match a target entropy, eliminating manual exploration tuning. A comprehensive description of TD3 is given in Algorithm 2.

Summary. DDPG introduces deterministic off-policy gradients but relies on ad-hoc action noise for exploration and is fragile. TD3 remedies over-estimation and variance with twin critics, smoothed targets, and delayed actor steps, still using simple Gaussian exploration noise. SAC switches

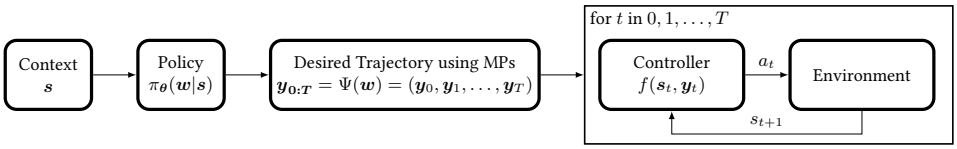


Figure 2.5.: Prediction and Environment Interaction of Episodic RL [10].

to stochastic actors, couples twin critics with entropy regularisation, and learns the exploration temperature automatically, offering state-of-the-art robustness and performance in continuous-control tasks.

2.2.7. Episodic Reinforcement Learning (ERL)

Unlike step-based RL (SRL) methods such as PPO [7] or SAC [8], which optimize a policy by boot-strapping from per-step value estimates, *episodic RL* searches directly in the space of *whole trajectories*. The learning signal is the return accumulated over an entire episode, making ERL naturally suited to sparse or non-Markovian reward settings [9, 90, 91].

Early black-box approaches. The first ERL algorithms applied evolutionary or other black-box optimization to the weights of small multilayer perceptrons [90, 91, 92]. With modern compute, scalable evolution strategies have been shown to match SRL on high-dimensional locomotion tasks, albeit with far larger sample budgets [93, 94].

Movement-primitive policies. To cut the search space from thousands of network parameters to a few dozen coefficients, Peters et al. [9] proposed representing policies by movement primitives (MPs) [1, 3]. MP-based ERL has achieved impressive results in robot baseball [9], juggling [95], and dexterous manipulation [96]. Model-based extensions further improve sample efficiency by evaluating candidate trajectories in a learned dynamics model [62]. Contextual variants handle changing goals via linear or neural policies conditioned on task parameters [10, 97, 98]. Recent work also explores safe or hierarchical skill acquisition within the episodic framework [41, 99].

As shown in Fig. 2.5, ERL predicts the trajectory parameters distribution as $\pi(\boldsymbol{w}|\boldsymbol{s})$, which shifts the solution search from the per-step action space $\mathcal{A}$ to the parameter space $\mathcal{W}$. Then, a parameter vector is sampled and translated into a whole desired trajectory. A controller is used to track the trajectory and interact with the environment. The per-step action $\boldsymbol{a}_t$ can be recognized either as the *per-step desired trajectory*, i. e. $\boldsymbol{a}_t = \boldsymbol{y}_t$ or the *low-level control command* i. e. $\boldsymbol{a}_t = f(\boldsymbol{s}_t, \boldsymbol{y}_t)$. Fig. 2.5 uses the latter option.

In contrast to SRL, a trajectory parameterized by a vector $\boldsymbol{w}$ is typically treated as a single data point in $\mathcal{W}$. Consequently, ERL commonly employs black-box optimization methods for problem-solving [10, 93]. The general learning objective of ERL is formally expressed as

$$J = \int \pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{s})[R(\boldsymbol{s},\boldsymbol{w}) - V^{\pi}(\boldsymbol{s})]d\boldsymbol{w} = \mathbb{E}_{\boldsymbol{w}\sim\pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{s})}[A(\boldsymbol{s},\boldsymbol{w})], \tag{2.55}$$

where $\pi_{\boldsymbol{\theta}}$ represents the policy, parameterized by $\boldsymbol{\theta}$, e. g. using NNs. The initial state $\boldsymbol{s} \in \mathcal{S}$ characterizes the starting configuration of the environment and the task goal, serving as the input to the policy. The $\pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{s})$ indicates the likelihood of selecting the trajectory parameter $\boldsymbol{w}$. The term $R(\boldsymbol{s},\boldsymbol{w}) = \sum_{t=0}^{T}[\gamma^t r_t]$ represents the return obtained from executing the trajectory, while $V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{w}\sim\pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{s})}[R(\boldsymbol{s},\boldsymbol{w})]$ denotes the expected return across all possible trajectories under policy $\pi_{\boldsymbol{\theta}}$. Their subtraction is defined as the advantage function $A(\boldsymbol{s},\boldsymbol{w})$, which quantifies the benefit of selecting a specific trajectory. By using parameterized trajectory generators like MPs, ERL benefits from consistent exploration, smooth trajectories, and robustness against local optima, as noted by [10].

Limitations and research gap. Episodic methods pay for their trajectory-level elegance in two ways. First, one episode becomes *one data point*: an update discards all intermediate state transitions that step-based RL exploits via temporal-difference (TD) learning. Most ERL algorithms therefore fall back on black-box search over a low-dimensional parameter space—evolutionary strategies [100], cross-entropy methods [101], or movement-primitive weights—and incur much larger sample budgets than SRL counterparts [10, 93, 98, 102]. Second, almost every ERL variant is strictly *on-policy*. Past episodes cannot be reused because of the challenges in learning off-policy critic for whole trajectories. In contrast, efficient SRL methods (e.g. SAC) rely on TD targets that assume an action is chosen at every visited state [8, 103], leading to simpler Q-function learning. Taken together, the *one-trajectory = one-datum* constraint and the absence of an off-policy critic leave current ERL techniques markedly less sample-efficient than modern step-based RL. These challenges will be addressed by this dissertation in Chapter 4 and 5.

2.3. Other Techniques Utilized in the Current Dissertation

2.3.1. Gaussian Parameterization via Neural Networks

Diagonal and Cholesky-Based Parameterization. A common way to represent a multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ in neural networks is to learn the mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$ using separate output heads. For computational efficiency, many implementations assume $\boldsymbol{\Sigma}$ to be diagonal. In this case, the network outputs the mean vector $\boldsymbol{\mu}$ and a log-standard deviation vector $\log \boldsymbol{\sigma}$, which is exponentiated to obtain the positive diagonal entries of $\boldsymbol{\Sigma}$. This representation enables stable training but cannot capture correlations between dimensions.

To model correlations, a full or structured covariance matrix is required. A widely adopted method is the Cholesky decomposition [104] $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^\top$, where $\boldsymbol{L}$ is a lower-triangular matrix. This parameterization ensures that $\boldsymbol{\Sigma}$ is positive definite [105]. However, the Cholesky factor is not unique, as multiple $\boldsymbol{L}$ matrices can yield the same $\boldsymbol{\Sigma}$. To resolve this ambiguity and ensure consistent optimization, the diagonal elements of $\boldsymbol{L}$ are constrained to be strictly positive. This is achieved using the *softplus* transformation [106],

$$\text{softplus}(x) = \log(1 + e^x), \tag{2.56}$$

which maps real-valued inputs to positive outputs in a smooth and differentiable manner.

In practice, to prevent the resulting diagonal entries from becoming too close to zero—an issue that can lead to numerical instability when inverting $\boldsymbol{\Sigma}$ or computing the log-determinant during likelihood evaluation—a small constant ϵ is added:

$$\boldsymbol{L}_{ii} = \text{softplus}(x_i) + \epsilon, \tag{2.57}$$

where ϵ is typically set to a value like 10^{-5} or 10^{-4}. This lower bound ensures that $\boldsymbol{\Sigma}$ remains well-conditioned and invertible, which is crucial for applications such as Gaussian likelihood computation and entropy-regularized policy gradients.

Differentiable Sampling via the Reparameterization Trick. To enable backpropagation through stochastic sampling from Gaussian distributions, the reparameterization trick is employed [107, 108]. This technique reformulates the sampling process as a deterministic and differentiable transformation of a noise variable, allowing gradients to flow through the sampled variables during optimization.

For a diagonal Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2\boldsymbol{I})$, where the covariance matrix is diagonal, the reparameterization is straightforward:

$$\boldsymbol{x} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \tag{2.58}$$

where $\odot$ denotes element-wise multiplication. Here, $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are outputs of the neural network, and $\boldsymbol{\epsilon}$ is a standard normal noise vector that is independent of the network parameters. This formulation preserves gradient flow through $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$, enabling efficient stochastic optimization, such as in variational inference and stochastic policy learning.

For a full covariance Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, the reparameterization is slightly more involved. Given the Cholesky decomposition $\boldsymbol{\Sigma} = \boldsymbol{L}\boldsymbol{L}^\top$, the sample is generated as:

$$\boldsymbol{x} = \boldsymbol{\mu} + \boldsymbol{L}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}). \tag{2.59}$$

In this case, the matrix-vector product $\boldsymbol{L}\boldsymbol{\epsilon}$ introduces correlated noise aligned with the geometry of the covariance structure. Since $\boldsymbol{L}$ is computed via differentiable transformations of the network outputs (e.g., softplus-activated diagonals), the entire sampling process remains differentiable with respect to the network parameters.

This reparameterized sampling is essential in gradient-based optimization involving probabilistic models, including variational autoencoders [107, 108], stochastic policies [8], and uncertainty-aware trajectory models. It enables low-variance, unbiased gradient estimation while capturing both the mean and the structure of uncertainty.

Gradient Behavior and the Role of Entropy. In many reinforcement learning and probabilistic modeling settings, the Gaussian distribution is optimized via the negative log-likelihood (NLL) loss:

$$\mathcal{L}_{\text{NLL}} = -\log \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}) + \frac{1}{2}\log|\boldsymbol{\Sigma}| + C, \tag{2.60}$$

where C is a constant independent of the parameters. This loss naturally arises in maximum likelihood estimation, supervised learning, and entropy-regularized reinforcement learning [109].

We now consider the gradients of this loss with respect to the distribution parameters. For simplicity, we first analyze the diagonal case where $\boldsymbol{\Sigma} = \text{diag}(\boldsymbol{\sigma}^2)$. The gradient with respect to the mean is:

$$\nabla_{\boldsymbol{\mu}} \mathcal{L}_{\text{NLL}} = -\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}), \tag{2.61}$$

and the gradient with respect to the variance is:

$$\nabla_{\boldsymbol{\sigma}^2} \mathcal{L}_{\text{NLL}} = \frac{1}{2}\left[\boldsymbol{\sigma}^{-2} - (\boldsymbol{x}-\boldsymbol{\mu})^2 \odot \boldsymbol{\sigma}^{-4}\right], \tag{2.62}$$

where all operations are element-wise. These expressions reveal that the gradient magnitudes are inversely proportional to the variance. In the full covariance case, a similar structure holds:

$$\nabla_{\boldsymbol{\Sigma}} \mathcal{L}_{\text{NLL}} = \frac{1}{2}\left[\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1} - \boldsymbol{\Sigma}^{-1}\right], \tag{2.63}$$

again indicating that small values in $\boldsymbol{\Sigma}$ amplify gradient magnitudes.

This inverse relationship between the gradient and the scale of the variance or covariance can lead to instability in policy optimization. As the entropy of the distribution decreases—i.e., as $\boldsymbol{\Sigma}$ becomes smaller—the gradient norms increase. This feedback loop can cause aggressive and unstable updates [110], especially when entropy is not explicitly regularized.

To monitor and control this behavior, the entropy of the Gaussian distribution is an important diagnostic metric. For a diagonal Gaussian, the entropy is:

$$\mathcal{H}(\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2 \boldsymbol{I})) = \frac{1}{2}\sum_i \log(2\pi e \sigma_i^2), \tag{2.64}$$

which decreases logarithmically with $\boldsymbol{\sigma}^2$. In reinforcement learning, entropy is often encouraged via explicit entropy regularization or constrained entropy optimization to promote exploration and maintain stable gradients. Tracking entropy during training serves as a proxy for policy expressiveness and a safeguard against overly deterministic behavior.

Using Gaussian Policies in RL. In practice, most policy gradient algorithms adopt a diagonal Gaussian distribution for computational simplicity and numerical stability. Full-covariance models are rarely used due to the increased parameter count, the cost of sampling and log-determinant computation, and the risk of overfitting. Moreover, the variance is often implemented as *state-independent*, i.e., shared across all states, especially in methods like PPO and TRPO [65]. This reduces the risk of vanishing exploration in certain regions of the state space and has been shown to contribute to training stability. On the other hand, state-dependent variance is used in methods like SAC [8], where the expressiveness of the policy is crucial for handling diverse reward landscapes or continuous control environments.

While state-dependent variance can improve performance in some settings, it also introduces sensitivity to initialization and optimization hyperparameters. As reported in [65], seemingly minor choices in policy architecture or noise modeling can lead to large differences in performance and reproducibility, underlining the importance of principled design choices when parameterizing Gaussian policies in RL.

2.3.2. Bayesian Information Aggregation

Motivation and Limitations of Mean Aggregation. In many structured prediction and decision-making tasks, it is necessary to infer a global latent variable from multiple partial observations or contexts. In Neural Processes (NPs) [111], this is commonly achieved by encoding each context point into a latent representation and then applying a permutation-invariant operation—typically mean pooling—across all representations. While simple and efficient, this strategy assumes all context points are equally informative, and it fails to reflect their varying levels of certainty or redundancy.

Such limitations are especially problematic when dealing with heterogeneous, sparse, or conflicting observations. In these cases, aggregation mechanisms should ideally reflect the relative confidence in each observation and support a probabilistically grounded combination rule.

Bayesian Aggregation via Product of Experts. Volpp et al. [17] address this challenge by proposing a Bayesian aggregation mechanism for combining local latent distributions. Each context point $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ is encoded into a latent Gaussian expert:

$$q_i(z) = \mathcal{N}(z \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i), \tag{2.65}$$

where $\boldsymbol{\mu}_i$ and $\boldsymbol{\Sigma}_i$ are the mean and covariance estimated from the context encoder. These experts represent local beliefs over the shared latent variable z.

To aggregate these into a global posterior, a product-of-experts (PoE) [112] formulation is used:

$$q(z \mid \mathcal{C}) \propto \prod_{i=1}^{n} q_i(z) \propto \mathcal{N}(z \mid \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*), \tag{2.66}$$

where the resulting mean and covariance have closed-form expressions:

$$\boldsymbol{\Sigma}_*^{-1} = \sum_{i=1}^{n} \boldsymbol{\Sigma}_i^{-1}, \quad \boldsymbol{\mu}_* = \boldsymbol{\Sigma}_* \left(\sum_{i=1}^{n} \boldsymbol{\Sigma}_i^{-1} \boldsymbol{\mu}_i \right). \tag{2.67}$$

This aggregation mechanism naturally assigns more weight to context points with lower uncertainty (i.e., smaller covariance), allowing reliable observations to dominate the posterior while down-weighting noisy or ambiguous ones.

Benefits and Practical Considerations. Compared to mean aggregation, this method provides an uncertainty-aware posterior that is analytically consistent and sample-efficient. Importantly, it preserves permutation invariance while overcoming the underfitting problems often associated with uniform averaging in NPs [113]. Moreover, the closed-form aggregation enables efficient inference and backpropagation, making it compatible with standard end-to-end training.

Volpp et al. also demonstrate that this approach outperforms traditional NPs in settings with highly variable context quality, such as meta-regression with sparse data. When combined with stochastic encoders and expressive decoders, Bayesian aggregation provides a robust inference mechanism suitable for tasks such as probabilistic skill learning, trajectory estimation, and policy conditioning—where balancing uncertainty across sources is crucial.

Usage in This Dissertation. In this dissertation, Bayesian aggregation is used to in Chapter 3, to combine multiple partial latent representations extracted from temporally segmented or spatially distributed observations. Rather than collapsing them via mean pooling, the use of PoE allows the model to assign higher influence to consistent and informative segments while discounting noisy or low-entropy ones. This yields improved generalization, more calibrated uncertainty, and better gradient flow in downstream tasks.

2.3.3. Transformer Architectures for Sequence Modeling

The Transformer architecture [11] has emerged as a powerful sequence modeling paradigm that replaces recurrence with self-attention mechanisms. Its design enables effective processing of temporally structured data and is particularly suited for sequence-to-sequence (seq2seq) tasks, such as machine translation, text summarization, and, more recently, policy learning in reinforcement learning.

At its core, the Transformer models the conditional sequence distribution $p(\mathbf{y}|\mathbf{x})$ by encoding an input sequence $\mathbf{x} = (x_1, x_2, ..., x_T)$ into a sequence of latent representations, and then decoding an output sequence $\mathbf{y} = (y_1, y_2, ..., y_{T'})$ autoregressively. Unlike recurrent models (e.g., LSTMs [114] or GRUs [115]), Transformers process the entire sequence in parallel, enabling significantly higher throughput and better modeling of long-range dependencies.

Self-Attention Mechanism. The central operation in the Transformer is the scaled dot-product self-attention. Given an input matrix $X \in \mathbb{R}^{T \times d}$, each element is projected into a triplet of queries $Q = XW^Q$, keys $K = XW^K$, and values $V = XW^V$, where $W^Q, W^K, W^V \in \mathbb{R}^{d \times d_k}$. The attention output is computed as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right)V. \tag{2.68}$$

This mechanism allows each element to attend to all others in the sequence, enabling the model to capture global temporal dependencies without positional bias.

Multi-Head Attention. To enrich representation capacity, the Transformer employs multi-head attention by projecting inputs into h separate query-key-value spaces and concatenating their attention outputs:

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O, \tag{2.69}$$

where each head is computed as $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$.

Positional Encoding. Unlike recurrent models that naturally encode order through their sequential computation, Transformers operate on sets and require explicit mechanisms to inject temporal information. The original Transformer [11] used fixed sinusoidal functions to define position encodings, enabling extrapolation to unseen sequence lengths. However, fixed encodings lack task-specific adaptability. To address this, many modern architectures (e.g., BERT [116], GPT [117]) adopt *learned positional embeddings*, where a trainable embedding vector is associated with each position index. These models demonstrate better empirical performance, particularly in data-rich regimes. Further advancements explore relative positional encodings [118, 119], where the attention computation is modified to incorporate pairwise position offsets. This improves generalization in tasks where relative distance is more important than absolute position, such as in temporal alignment or robotic trajectories. Rotary position embeddings (RoPE) [120] go a step further by integrating relative position information directly into the query and key vectors using complex rotations, allowing seamless extrapolation and compatibility with parallel attention computation.

In robotic applications and reinforcement learning, relative and rotary encodings are particularly appealing, as they emphasize invariance to absolute timestep indices while capturing motion-dependent dependencies more effectively.

Architectural Components. Each encoder and decoder block in a Transformer, shown in Fig. 2.6, contains several key sub-layers:

- **Multi-head attention:** enables content-based interaction across time steps by computing multiple attention distributions in parallel, enhancing the model's ability to represent diverse relationships.

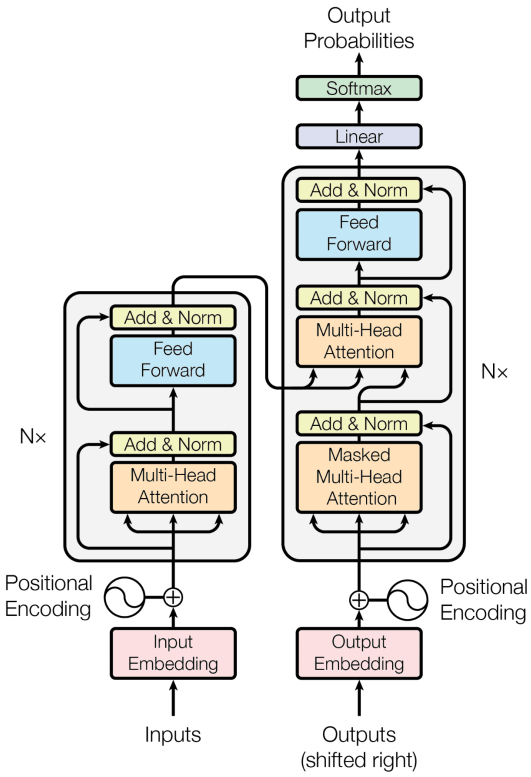


Figure 2.6.: An encoder - decoder transformer architecture, originally from Vaswani [11].

- **Feed-forward network (FFN):** applies a position-wise multi-layer perceptron to each token embedding independently. The canonical FFN has two linear layers with a ReLU [121] or GELU [122] activation in between.
- **Layer normalization [123] and residual connections[124]:** stabilize training and facilitate gradient propagation. Each sub-layer is wrapped with a residual connection followed by layer normalization: LayerNorm(x + Sublayer(x)), which normalizes the hidden activations to zero mean and unit variance.

While the original Transformer [11] uses *pre-norm residual blocks*, many practical implementations such as GPT [117] and BERT [116] adopt *post-norm* [125] variants where layer normalization is applied before the sub-layer (i.e., Sublayer$(x + \text{LayerNorm}(x))$). Post-norm was initially found to improve convergence in deep models, but recent studies favor pre-norm for better training stability in very deep architectures.

Modern Transformers also often increase the *FFN dimensionality* (e.g., using a 4x or even 8x expansion factor relative to the model dimension) and replace ReLU with *GELU activations* for smoother gradients [122]. Additionally, *Dropout* [126], *residual scaling* [127], and *attention dropout* are commonly used for regularization.

Architectural extensions such as *sparse attention* [128], *linear attention* [129], and *low-rank adapters* [130] have been proposed to improve scalability and memory efficiency. These advances are particularly beneficial in long-horizon or resource-constrained reinforcement learning settings.

Overall, these architectural refinements contribute to the wide adoption of Transformer variants in large-scale learning systems, including language models [117], vision transformers [131], and policy learning in robotics [132].

Advantages over RNN-based Models. Transformers provide several key advantages compared to traditional RNN-based architectures:

- **Parallel computation:** Unlike RNNs [133], which process inputs sequentially, Transformers operate on entire sequences in parallel, significantly accelerating training and making them more suitable for large-scale datasets.
- **Long-range dependency modeling:** Self-attention enables direct interactions between all token pairs, regardless of distance, whereas RNNs suffer from vanishing gradients and degraded performance over long sequences.
- **Flexible context aggregation:** Transformers can selectively attend to relevant positions across the sequence, while RNNs compress context into a single hidden state, limiting their representational capacity.
- **Stable optimization:** The use of layer normalization and residual connections in Transformers improves gradient flow and convergence stability, which are known challenges in training deep or recurrent models.

Transformers in model-free RL. Early study on applying sequence models in model-free RL mainly focused on solving *Partial Observable Markov Decision Processes* (POMDPs) [45] in online RL settings. In these studies, RNN-based models, such as LSTMs and GRUs, were predominantly used [134, 135, 136, 137, 138]. Inspired by the success of Transformers in domains requiring sequence reasoning, the study incorporating Transformers in RL to solve tasks that require long-horizon memory emerged. However, using standard Transformers in RL could results in performance comparable to random policy [139]. To address this issue, *Gated Transformer-XL* (GTrXL) [139] augmented Transformer-XL with GRU-style gating layers between multi-head self-attention layers, stabilizing the training of deep Transformer networks (up to 12 layers) with online RL. Another research line focuses on utilizing Transformers to enhance offline RL, where the learning process is based on a fixed dataset collected by arbitrary behavior policies.

Decision Transformers [132] were the first to formulate offline RL as a sequence modeling problem. Subsequent works extended this approach by incorporating dynamic history length adjustment [140], Q-learning [141], and replacing the Transformer with a more efficient state-space model [142]. Online Decision Transformers [143] further advanced Decision Transformer by introducing online fine-tuning. In contrast to these studies, which primarily focus on offline RL or fine-tuning pre-trained models, the proposed method in this dissertation, TOP-ERL [14], is designed for online RL and does not rely on offline training. Additionally, TOP-ERL is not designed to solve tasks that require long-horizon memory. Instead, it focuses on using a Transformer-based critic to improve multi-step TD learning within the ERL framework.

3. Unifying Dynamic and Probabilistic Movement Primitives

Existing Movement Primitive (MP) methods usually tackle only one side of the problem: either they guarantee smooth, stable dynamics or they capture rich trajectory statistics through probabilistic modelling. This separation limits expressiveness, because real-world robotic skills demand both properties at once. Bridging this gap motivates **[RQ1]**: *How can dynamical smoothness and probabilistic trajectory modelling be integrated within a single MP framework?*

Contributions. To answer this question we introduce *Probabilistic Dynamic Movement Primitives (ProDMPs)*, a unified MP formulation that combines the key advantages of dynamical and probabilistic approaches.

1. *Closed-form dynamics.* By analytically solving the ordinary differential equations of classical DMPs [1], ProDMPs replace costly numerical integration with pre-computed position and velocity basis functions. This yields fast trajectory generation from arbitrary initial conditions while preserving the smoothness and adaptability characteristic of dynamical MPs.

2. *Probabilistic layer.* Building on [3], ProDMPs map distributions in parameter space directly into trajectory space, capturing temporal and cross-DoF correlations and enabling uncertainty-aware replanning. Smooth transitions are maintained even when trajectory parameters are updated online.

3. *Deep sensory conditioning.* Embedding ProDMPs in a deep network with Bayesian Aggregation [17] allows conditioning on high-dimensional inputs such as images [33]. The network learns non-linear mappings from sensory context to ProDMP parameters, further expanding the framework's flexibility.

Comprehensive experiments—including vision-guided digit writing and object-aware manipulation—show that ProDMPs deliver smoother trajectories, higher representational accuracy, and markedly lower computational cost than prior MP formulations, establishing them as a solid foundation for unified dynamical–probabilistic skill learning.

The remainder of this chapter has been published as [12]: Ge Li, Zeqi Jin, Michael Volpp, Fabian Otto, Rudolf Lioutikov, and Gerhard Neumann. "ProDMP: A Unified Perspective on Dynamic and Probabilistic Movement Primitives." *In: IEEE Robotics and Automation Letters (RAL),* 2023.

3.1. Introduction

Movement Primitives MPs are a prominent tool for motion representation and synthesis in robotics. They serve as basic movement elements, modulate the motion behavior, and form more complex movements through combination or concatenation. This work focuses on trajectory-based movement representations [1, 3]. Given a parameter vector, such representations generate desired trajectories for the robot to follow. These methods have gained much popularity in imitation and reinforcement learning (IL, RL) [144, 16, 98, 10] due to their concise parametrization and flexibility. Current methods can be roughly classified into approaches based on dynamical systems [1, 2, 33, 22, 15] and probabilistic approaches [3, 145, 4], with both types offering their own advantages. The dynamical systems-based approaches, such as Dynamic Movement Primitives (DMPs), guarantee that the generated trajectories start precisely at the current position and velocity of the robot, which allows for smooth trajectory replanning i. e., changing the parameters of the MPs during motion execution [15, 146]. However, since DMPs represent the trajectory via the forcing term instead of a direct representation of the position, numerical integration from acceleration to position has to be applied iteratively, which constitutes an additional workload and makes the trajectory statistics estimation difficult [147]. Probabilistic methods, such as Probabilistic Movement Primitive (ProMP), are able to acquire such statistics, thus making them the key enablers for acquiring variable-stiffness controllers and the trajectory's temporal and DoF correlation. These methods further perform as generative models, facilitating the sampling of new trajectories. However, the lack of internal dynamics of these approaches suffers from discontinuities in position and velocity between old and new trajectories in the case of replanning.

In this work, we propose Probabilistic Dynamic Movement Primitives (ProDMPs) which unify both methodologies. We show that the trajectory of a DMP, obtained by integrating its second-order dynamical system, can be expressed by a linear basis function model that depends on the parameters of the DMP, i. e., the weights of the forcing function and the goal attractor. The linear basis functions can be obtained by integrating the original basis functions used in the DMP - an operation that only needs to be performed once offline in the ProDMPs. Recently, MP research has been extended to deep neural network (NN) architectures [22, 15, 4] that enable conditioning the trajectory on high-dimensional inputs, e. g., images. Following these ideas, we embed ProDMPs into a deep architecture that allows non-linear conditioning on a varying number of conditioning events. These events are aggregated using Bayesian Aggregation (BA) into a latent probabilistic representation [17] which is mapped to a Gaussian distribution in the parameter space of the ProDMPs.

We summarize the contributions of this work as:

1. We unify ProMPs and DMPs into one consistent framework that inherits the benefits of both
2. We enable to compute distributions and to capture correlations of DMPs trajectories
3. The robot's current state can be inscribed into the trajectory distribution through initial conditions, allowing for smooth replanning

4. The offline computation of the position and velocity basis functions significantly reduces the workload of NN architectures and computation time by a factor of 10
5. We embed ProDMPs in a deep encoder-decoder architecture that allows non-linear conditioning on high-dimensional observations with varying information levels

We evaluate our method on three digit-writing tasks using images as inputs, a simulated robot pushing task with a complex physical interaction, and a real robot picking task with shifting object positions. We compare our model with state-of-the-art NN-based DMPs [33, 22, 15] and the NN-based probabilistic method [4].

3.2. Related Work

Paraschos et al. [3] established ProMPs to model MPs as a trajectory distribution that captures temporal and DoF correlation. ProMPs use a Gaussian distribution over the parameters and map it to the corresponding trajectory distribution using a linear basis function model. In contrast, such a mapping is not allowed for the DMP-based approaches. Previous methods, like GMM/GMR-DMPs [23, 24] used Gaussian Mixture Models to cover the trajectories' domain. Yet, this does not capture temporal correlation nor provide a generative approach to trajectories. Other methods which have learned distributions over DMP weights [25, 26], do not connect the weights distribution to the trajectory distribution, as trajectories can only be obtained by integration. Hence, it is also hard to learn the weights distribution reversely from trajectories in an end-to-end manner. Methods like [148, 149] model the motion's statistics using a *state-dependent* dynamic system and directly learn the parameters of the governing differential equation from the trajectory. However, these methods have difficulties in modeling highly-dynamic time-dependent motions and correlations as well as scaling the execution speed of these motions. Moreover, it is not straightforward to integrate high-dimensional sensory observations in these architectures. To learn DMP parameters from high-dimensional sensory inputs, [33, 22] designed an encoder-decoder architecture to learn a single DMP from digit images, and derive the gradient of the trajectory with respect to the learnable parameters. Bahl et al. [15] propose Neural Dynamic Policies (NDP) that allow replanning the DMP parameters throughout the trajectory execution in both IL and RL settings. The learning objective of these two methods for IL is to optimize the mean squared error (MSE) between the predicted and the ground-truth trajectories using back-propagation. However, to formulate a trajectory, DMPs must apply numerical integration in an iterative manner during the NN training [33, 22, 15], which significantly increases the computational workload, rendering these approaches cumbersome to use. Additionally, the integration-based trajectory representation limits the use of probabilistic methods and hence these NN-DMPs approaches cannot be trained using a log-likelihood (LL) loss.

Probabilistic MPs methods have also been extended with deep architectures. Seker et al. [4] directly use a Conditional Neural Processes model [34] as a trajectory generator, i. e., Conditional Neural Movement Primitives (CNMPs), to predict the trajectory distribution. While such a model enables non-linear conditioning on high-dimensional inputs, it can only predict an isotropic trajectory variance at each time step. The temporal and DoF correlations are missing, which makes sampling

consistently in time and DoF infeasible. Besides, both ProMPs and CNMPs neglect dynamics, i. e. , when changing trajectory parameters in run-time, the newly generated trajectory will jump at the replanning time point. To execute such trajectories, a heuristic controller is used to freeze the time and catch up with the jump [4]. However, such a waiting mechanism does not scale to time-sensitive motions and tasks.

3.3. A Unified Perspective on Dynamic and Probabilistic Movement Primitives

We first briefly cover the fundamental aspects of DMPs. Then, we derive the analytical solution of the DMPs' ODE to develop our new ProDMPs representation. For convenience, we introduce our approach through a 1-DoF dynamical system and later extend it to a high-DoF system.

3.3.1. Solving DMPs' Ordinary Differential Equation

For a single movement execution as a trajectory $\boldsymbol{y}_{0:T} = [y(0),\ y(1),\ \ldots,\ y(T)]$, Schaal [1], Ijspeert et al. [2] model it as a second-order linear dynamical system with a non-linear forcing function f,

$$\tau^2 \ddot{y} = \alpha(\beta(g - y) - \tau \dot{y}) + f(x), \quad f(x) = x \frac{\sum_m \varphi_m(x) w_m}{\sum_m \varphi_m(x)} = x \boldsymbol{\varphi}_x^\top \boldsymbol{w}, \qquad \text{(from Eq. 2.1)}$$

where $y = y(t),\ \dot{y} = \mathrm{d}y/\mathrm{d}t,\ \ddot{y} = \mathrm{d}^2 y/\mathrm{d}t^2$ denote the position, velocity, and acceleration of the system at a specific time $t \in [0, T]$, respectively. Constants α and β are spring-damper parameters, g represents the goal attractor of the system when $t \to \infty$, and τ is a time constant that modulates the speed of trajectory execution. To ensure convergence towards the goal, DMPs employ a forcing function governed by an exponentially decaying phase variable $x(t) = \exp(-\alpha_x/\tau; t)$. In Eq. 2.1, $\varphi_i(x)$ represents the basis functions for the forcing term. The trajectory's shape as it approaches the goal is determined by the weight parameter vector $\boldsymbol{w} = [w_1,\ \ldots,\ w_M]$. The trajectory $\boldsymbol{y}_{0:T}$ is typically obtained by numerically integrating the dynamical system from $t = 0$ to $t = T$ [22, 15]. Starting from the initial position y_0 and velocity $\dot{y}_0$, *Euler integration* proceeds iteratively:

$$\begin{aligned}
&\textbf{Acceleration:} && \ddot{y}_t = \frac{1}{\tau^2}[\alpha(\beta(g - y_{t-1}) - \tau \dot{y}_{t-1}) + f(x(t-1))], \\
&\textbf{Velocity:} && \dot{y}_t = \dot{y}_{t-1} + \ddot{y}_{t-1}\mathrm{d}t, \\
&\textbf{Position:} && y_t = y_{t-1} + \dot{y}_{t-1}\mathrm{d}t.
\end{aligned} \qquad \text{(from Eq. 2.2)}$$

The dynamical system defined in Eq. (2.1) is a second-order linear non-homogeneous ordinary differential equation (ODE) with constant coefficients, whose closed-form solution can be derived analytically. We rewrite Eq. (2.1) and its homogeneous counterpart in standard form as

$$\textbf{DMPs' ODE:} \qquad \ddot{y} + \frac{\alpha}{\tau}\dot{y} + \frac{\alpha\beta}{\tau^2}y = \frac{f(x)}{\tau^2} + \frac{\alpha\beta}{\tau^2}g \equiv F(x, g), \tag{3.1}$$

$$\textbf{Homo. ODE:} \qquad \ddot{y} + \frac{\alpha}{\tau}\dot{y} + \frac{\alpha\beta}{\tau^2}y = 0, \tag{3.2}$$

where F denotes some function of x and g. Using the method of variation of constants [150] the closed-form solution of the second-order ODE in Eq. (3.1), i. e., the trajectory position, is

$$\textbf{Analytical form, position:} \qquad y = c_1 y_1 + c_2 y_2 - y_1 \int \frac{y_2 F}{\eta}\mathrm{d}t + y_2 \int \frac{y_1 F}{\eta}\mathrm{d}t, \tag{3.3}$$

where y_1, y_2 are two linearly independent complementary functions of the homogeneous ODE given in Eq. (3.2). c_1, c_2 are two constants which are determined by the initial conditions of the ODE, and $\eta = y_1\dot{y}_2 - \dot{y}_1 y_2$. Both integrals in Eq. (3.3) are indefinite. With appropriate values $\beta = \alpha/4$, the system is critically damped [1, 2], and the corresponding characteristic equation of the homogeneous ODE, i. e., $\Delta = (\alpha^2 - 4\alpha\beta)/\tau^2$ will be 0. Consequently, y_1, y_2 will take the form

$$\textbf{Complementary func.:} \qquad y_1 = y_1(t) = \exp\left(-\frac{\alpha}{2\tau}t\right),\ y_2 = y_2(t) = t\exp\left(-\frac{\alpha}{2\tau}t\right). \tag{3.4}$$

Using this result, the term η can also be simplified to $\eta = \exp(-\alpha t/\tau) \neq 0$. To get y, we need to solve the two indefinite integrals in Eq. (3.3) as

$$\begin{aligned} I_1(t) &= \int \frac{y_2 F}{\eta}\mathrm{d}t = \int t\exp\left(\frac{\alpha}{2\tau}t\right)F(x, g)\mathrm{d}t, \\ I_2(t) &= \int \frac{y_1 F}{\eta}\mathrm{d}t = \int \exp\left(\frac{\alpha}{2\tau}t\right)F(x, g)\mathrm{d}t. \end{aligned} \tag{3.5}$$

Applying the Fundamental Theorem of Calculus, i. e., $\int h(t)\mathrm{d}t = \int_0^t h(t')\mathrm{d}t' + c$, $c \in \mathbb{R}$, together with the definition of the forcing function f in Eq. (2.1) and $F(x, g)$ in Eq. (3.1), $I_1(t)$ can be expressed as

$$I_1(t) = \frac{1}{\tau^2}\left[\int_0^t t'\exp\left(\frac{\alpha}{2\tau}t'\right)x(t')\boldsymbol{\varphi}_x^\top \boldsymbol{w}\mathrm{d}t' + \int_0^t t'\exp\left(\frac{\alpha}{2\tau}t'\right)\frac{\alpha^2}{4}g\mathrm{d}t'\right] + c_3 \tag{3.6}$$

$$= \frac{1}{\tau^2}\left[\int_0^t t'\exp\left(\frac{\alpha}{2\tau}t'\right)x(t')\boldsymbol{\varphi}_x^\top \mathrm{d}t'\right]\boldsymbol{w} + \left[\left(\frac{\alpha}{2\tau}t - 1\right)\exp\left(\frac{\alpha}{2\tau}t\right) + 1\right]g + c_3, \tag{3.7}$$

where c_3 is a constant fixed by the initial conditions. From Eq. (3.6) to Eq. (3.7), we move the time-independent parameters $\boldsymbol{w}$ and g out of their corresponding integrals. Notice that the remaining part of the second integral has an analytical solution. The remaining part of the first integral,

however, has no closed-form solution because the basis functions $\boldsymbol{\varphi}_x$ may be arbitrarily complex. Denoting these integrals as

$$\begin{aligned} \boldsymbol{p}_1(t) &\equiv \frac{1}{\tau^2}\int_0^t t' \exp\left(\frac{\alpha}{2\tau}t'\right) x(t')\boldsymbol{\varphi}_x^\top \mathrm{d}t', \\ q_1(t) &\equiv \left(\frac{\alpha}{2\tau}t - 1\right)\exp\left(\frac{\alpha}{2\tau}t\right) + 1, \end{aligned} \tag{3.8}$$

where $\boldsymbol{p_1}$ is a N-dim vector and q_1 a scalar, we can express $I_1(t) = \boldsymbol{p}_1(t)^\top \boldsymbol{w} + q_1(t)g + c_3$. Following the same steps, we can obtain a similar solution for I_2, i. e., $I_2(t) = \boldsymbol{p}_2(t)^\top \boldsymbol{w} + q_2(t)g + c_4$, where we present $\boldsymbol{p}_2(t)$ and $q_2(t)$ in Eq. (3.9).

$$\begin{aligned} \boldsymbol{p}_2(t) &\equiv \frac{1}{\tau^2}\int_0^t \exp\left(\frac{\alpha}{2\tau}t'\right) x(t')\boldsymbol{\varphi}_x^\top \mathrm{d}t', \\ q_2(t) &\equiv \frac{\alpha}{2\tau}\left[\exp\left(\frac{\alpha}{2\tau}t\right) - 1\right]. \end{aligned} \tag{3.9}$$

3.3.2. DMPs' Linear Basis Functions Representation.

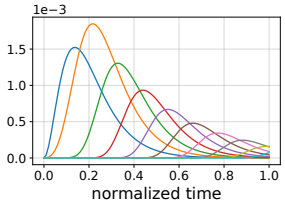
Substituting the two integrals I_1 and I_2 in Eq. (3.3) by their derived form, the constants c_3 and c_4 can then be merged into c_1 and c_2, respectively. We can now express the position of DMPs in Eq. (3.3) as a summation of complementary functions y_1 and y_2, plus a linear basis function representation of the weights $\boldsymbol{w}$ and the goal attractor g

$$\begin{aligned} y &= c_1 y_1 + c_2 y_2 + \begin{bmatrix} y_2\boldsymbol{p_2} - y_1\boldsymbol{p_1} & y_2 q_2 - y_1 q_1 \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ g \end{bmatrix} \\ &\triangleq\ c_1 y_1 + c_2 y_2 + \boldsymbol{\Phi}^\top \boldsymbol{w}_g, \end{aligned} \tag{3.10}$$

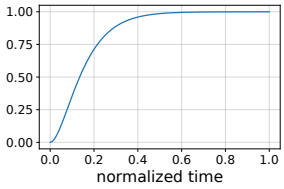
where $\boldsymbol{w}_g$ is a N+1-dim vector containing $\boldsymbol{w}$ and g. The resulting position basis functions for $\boldsymbol{w}$ and g are represented by $\boldsymbol{\Phi}(t)$, and its time derivative forms the velocity basis functions as $\dot{\boldsymbol{\Phi}}(t)$. These basis can be solved numerically and are shown in Fig. 3.1. The constants c_1 and c_2 are determined by solving a initial condition problem where we use the current position and velocity of the robot to inscribe where the trajectory should start.

3.3.3. Solving the Initial Condition Problem

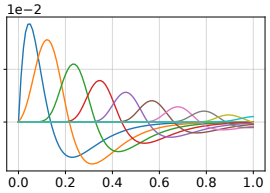
To compute the coefficients c_1 and c_2, we need to first obtain the velocity representation by computing the derivative of Eq. (3.3) w.r.t. time. The resulting equation reads



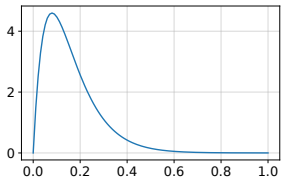
(a) Weights' position basis



(b) Goal's position basis



(c) Weights' velocity basis



(d) Goal's velocity basis

Figure 3.1.: The position basis functions $\boldsymbol{\Phi}$ and velocity basis functions $\dot{\boldsymbol{\Phi}}$ of the ProDMPs can be computed offline and later used in online trajectory computation.

$$\begin{aligned}\dot{y} &= c_1\dot{y}_1 + c_2\dot{y}_2 - \dot{y}_1 \int \frac{y_2 F}{\eta}\mathrm{d}t - y_1\frac{y_2 F}{\eta} + \dot{y}_2 \int \frac{y_1 F}{\eta}\mathrm{d}t + y_2\frac{y_1 F}{\eta} \\ &= c_1\dot{y}_1 + c_2\dot{y}_2 - \dot{y}_1 \int \frac{y_2 F}{\eta}\mathrm{d}t + \dot{y}_2 \int \frac{y_1 F}{\eta}\mathrm{d}t. \end{aligned} \tag{3.11}$$

Note that Eq. (3.3) and Eq. (3.11) share a similar structure using the same constants c_1 and c_2, as well as the two indefinite integrals in Eq. (3.5). The only difference is that the two complementary functions y_1 and y_2 are replaced by their derivatives $\dot{y}_1$ and $\dot{y}_2$:

$$\begin{aligned}\dot{y}_1 &= \dot{y}_1(t) = -\frac{\alpha}{2\tau}\exp\left(-\frac{\alpha}{2\tau}t\right), \\ \dot{y}_2 &= \dot{y}_2(t) = -\frac{\alpha}{2\tau}t\exp\left(-\frac{\alpha}{2\tau}t\right) + \exp\left(-\frac{\alpha}{2\tau}t\right). \end{aligned} \tag{3.12}$$

By reusing the derivation of Eq. (3.5, 3.7 ,3.8, 3.9), the trajectory velocity is ultimately derived as

$$\begin{aligned}\dot{y} =&\ c_1\dot{y}_1 + c_2\dot{y}_2 + \begin{bmatrix}\dot{y}_2\boldsymbol{p_2} - \dot{y}_1\boldsymbol{p_1} & \dot{y}_2 q_2 - \dot{y}_1 q_1\end{bmatrix}\begin{bmatrix}\boldsymbol{w} \\ g\end{bmatrix} \\ \triangleq&\ c_1\dot{y}_1 + c_2\dot{y}_2 + \dot{\boldsymbol{\Phi}}^\top \boldsymbol{w}_g, \end{aligned} \tag{3.13}$$

where $\dot{\boldsymbol{\Phi}}(t)$ represents the basis functions of the velocity. The position and velocity in Eq. (3.10, 3.13) share the same linear model structure, coefficients c_1, c_2, and the parameters $\boldsymbol{w}_g$.

We need two initial conditions to solve c_1 and c_2. Theoretically, there are three options,

1. two position values at two time steps, i. e., $y(t_{b_1}),\ y(t_{b_2}),\ t_{b_1} \neq t_{b_2}$,
2. two velocity values at two time steps, i. e., $\dot{y}(t_{b_1}),\ \dot{y}(t_{b_2}),\ t_{b_1} \neq t_{b_2}$, and
3. one position value plus one velocity value $y(t_{b_1}),\ \dot{y}(t_{b_2})$, where t_{b_1} and t_{b_2} are typically identical.

The third option allows us to naturally use the current position and velocity of a robot as initial conditions of a trajectory to be predicted. Consequently, our representation guarantees that the planned trajectory always satisfies any pair of position and velocity at a certain time step, if used as initial conditions. Such initial conditions do not have to take values at time step $t = 0$ but can be specified for any time step. To make the derivation clear, we recall the position and velocity representations of ProDMPs at one single time step, i.e.,

$$y = c_1 y_1 + c_2 y_2 + \boldsymbol{\Phi}^\top \boldsymbol{w}_g, \qquad \text{(from 3.10)}$$
$$\dot{y} = c_1 \dot{y}_1 + c_2 \dot{y}_2 + \dot{\boldsymbol{\Phi}}^\top \boldsymbol{w}_g. \qquad \text{(from 3.13)}$$

We denote the initial conditions as a position and velocity pair $(y_b, \dot{y}_b)$ at time step t_b. The values of the complementary functions in Eq. (3.4) and their corresponding derivatives at t_b are denoted by $y_{1_b}, y_{2_b}, \dot{y}_{1_b} \dot{y}_{2_b}$. We can now substitute these initial conditions and values into the position and velocity representations, and result in a binary linear equations system $\boldsymbol{A}\boldsymbol{c} = \boldsymbol{B} \rightarrow \boldsymbol{c} = \boldsymbol{A}^{-1}\boldsymbol{B}$, where in our case

$$\boldsymbol{A} = \begin{bmatrix} y_{1_b} & y_{2_b} \\ \dot{y}_{1_b} & \dot{y}_{2_b} \end{bmatrix},\ \boldsymbol{A}^{-1} = \frac{1}{\det \boldsymbol{A}} \begin{bmatrix} \dot{y}_{2_b} & -y_{2_b} \\ -\dot{y}_{1_b} & y_{1_b} \end{bmatrix},\ \boldsymbol{B} = \begin{bmatrix} y_b - \boldsymbol{\Phi}_b^\top \boldsymbol{w}_g \\ \dot{y}_b - \dot{\boldsymbol{\Phi}}_b^\top \boldsymbol{w}_g \end{bmatrix},\ \boldsymbol{c} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}. \quad (3.14)$$

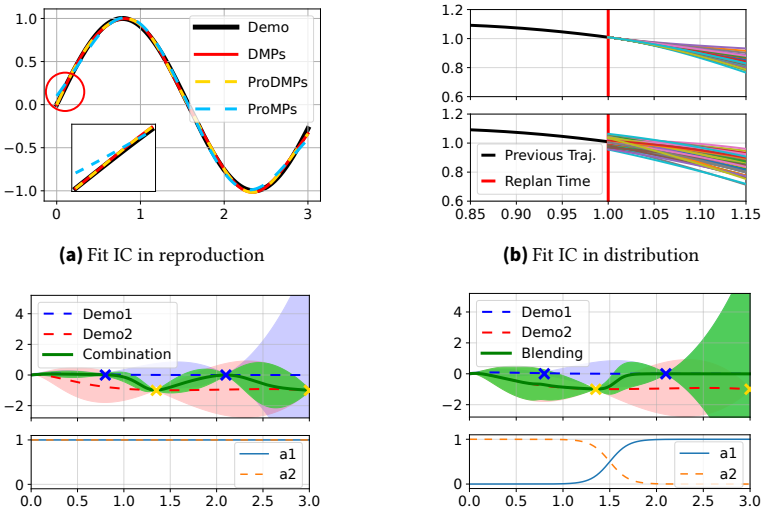
Since $\det \boldsymbol{A} = y_{1_b}\dot{y}_{2_b} - y_{2_b}\dot{y}_{1_b} = \exp\left(-\frac{\alpha}{\tau} t_b\right) \neq 0$, we can compute c_1, c_2 through these initial conditions terms, as well as the learnable parameters $\boldsymbol{w}_g$.

Init. cond. problem solution:

$$\begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} \frac{\dot{y}_{2_b} y_b - y_{2_b} \dot{y}_b}{y_{1_b}\dot{y}_{2_b} - y_{2_b}\dot{y}_{1_b}} + \frac{y_{2_b}\boldsymbol{\Phi}_b^\top - \dot{y}_{2_b}\boldsymbol{\Phi}_b^\top}{y_{1_b}\dot{y}_{2_b} - y_{2_b}\dot{y}_{1_b}} \boldsymbol{w}_g \\ \frac{y_{1_b}\dot{y}_b - \dot{y}_{1_b} y_b}{y_{1_b}\dot{y}_{2_b} - y_{2_b}\dot{y}_{1_b}} + \frac{\dot{y}_{1_b}\boldsymbol{\Phi}_b^\top - y_{1_b}\dot{\boldsymbol{\Phi}}_b^\top}{y_{1_b}\dot{y}_{2_b} - y_{2_b}\dot{y}_{1_b}} \boldsymbol{w}_g \end{bmatrix}. \quad (3.15)$$

Substituting Eq.(3.15) into Eq.(3.10) and (3.13), we can express the trajectory position and velocity at one time single step t as

$$y = \xi_1 y_b + \xi_2 \dot{y}_b + \left[\xi_3 \boldsymbol{\Phi}_b + \xi_4 \dot{\boldsymbol{\Phi}}_b + \boldsymbol{\Phi}\right]^\top \boldsymbol{w}_g, \quad (3.16)$$
$$\dot{y} = \dot{\xi}_1 y_b + \dot{\xi}_2 \dot{y}_b + \left[\dot{\xi}_3 \boldsymbol{\Phi}_b + \dot{\xi}_4 \dot{\boldsymbol{\Phi}}_b + \dot{\boldsymbol{\Phi}}\right]^\top \boldsymbol{w}_g, \quad (3.17)$$



(a) Fit IC in reproduction

(b) Fit IC in distribution

(c) Combination

(d) Blending

Figure 3.2.: ProDMP inherited the properties of DMPs and ProMPs, x-axis: time [s], y-axis: position [m]. (a) Given a demonstrated trajectory, ProDMPs can reproduce it and guarantee a fit of the boundary conditions (BCs), while ProMPs cannot. Besides, ProDMPs reproduce the identical trajectory of DMPs', given the same parameters $\boldsymbol{w}_g$. As our method uses the linear basis functions rather than online numerical integration, it is thus much faster. (b) ProDMPs (upper) can model trajectory distributions and guarantee that all newly sampled trajectories follow the given initial conditions, i.e., smooth online replanning. In contrast, ProMPs' samples (lower) cannot fit the initial condition well, which often results in a jump in the case of replanning. (c)+(d) ProDMPs support probabilistic operations of ProMPs, e.g., combination and blending [3] of two demonstrated trajectory distributions (blue and red) into one resulting distribution (green).

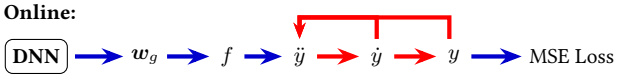
where

$$\begin{aligned} \xi_1 = \xi_1(t) &= \frac{\dot{y}_{2_b} y_1 - \dot{y}_{1_b} y_2}{y_{1_b}\dot{y}_{2_b} - y_{2_b}\dot{y}_{1_b}}, \qquad \xi_2 = \xi_2(t) = \frac{y_{1_b} y_2 - y_{2_b} y_1}{y_{1_b}\dot{y}_{2_b} - y_{2_b}\dot{y}_{1_b}}, \\ \xi_3 = \xi_3(t) &= \frac{\dot{y}_{1_b} y_2 - \dot{y}_{2_b} y_1}{y_{1_b}\dot{y}_{2_b} - y_{2_b}\dot{y}_{1_b}}, \qquad \xi_4 = \xi_4(t) = \frac{y_{2_b} y_1 - y_{1_b} y_2}{y_{1_b}\dot{y}_{2_b} - y_{2_b}\dot{y}_{1_b}}. \end{aligned} \tag{3.18}$$

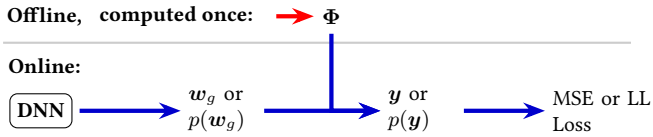
Replacing y_1, y_2 by $\dot{y}_1, \dot{y}_2$ respectively, we will get the derivatives $\dot{\xi}_k$. Eq.(3.16) and (3.17) show that the trajectory position y and velocity $\dot{y}$ are fully determined by the initial position y_b, and velocity $\dot{y}_b$ at the time step t_b, as well as the learnable weights $\boldsymbol{w}_g$.

3.3.4. Probability Distribution of Multi DoF DMPs

The linear basis function representation of ProDMPs takes the same form of ProMPs. Hence, our model can also be extended to multi-DoF systems $\boldsymbol{Y} = [y^1, \ldots, y^D]^\top$, where D denotes the system's DoF. Similar to [3] and recall Eq.(2.5), we extend the basis functions $\boldsymbol{\Phi}, \dot{\boldsymbol{\Phi}}$, as well as their



(a) NN + DMPs couple the numerical integration and the learning.



(b) NN + ProDMPs decouple the numerical integration from the learning pipeline and can also model trajectory distributions.

Figure 3.3.: Comparison of trajectory generation pipelines between (a) NN-based DMPs [33, 22, 15] and (b) ProDMPs. The node **DNN** represents arbitrary deep neural network architecture. The blue arrows denote the learning pipeline while the red arrows the numerical integration. Our method transforms the expensive numerical integration as basis functions computed offline which speeds up the trajectory computation and allows trajectory distribution prediction.

initial values $\boldsymbol{\Phi}_b, \dot{\boldsymbol{\Phi}}_b$ to block-diagonal matrices $\boldsymbol{\Psi}, \dot{\boldsymbol{\Psi}}, \boldsymbol{\Psi}_b, \dot{\boldsymbol{\Psi}}_b$ and concatenate each DoF's initial conditions into a vector, e. g. , $y_b^1, \ ..., \ y_b^D \ \rightarrow \boldsymbol{Y}_b = [y_b^1, \ ..., \ y_b^D]^\top$.

Additionally, we note that the coefficient constants c_1 and c_2 of each DoF can be solved independently. As a consequence, we have the multi-DoF trajectory as

$$\boldsymbol{Y} = \xi_1 \boldsymbol{Y}_b + \xi_2 \dot{\boldsymbol{Y}}_b + [\xi_3 \boldsymbol{\Psi}_b + \xi_4 \dot{\boldsymbol{\Psi}}_b + \boldsymbol{\Psi}]^\top \boldsymbol{w}_g, \tag{3.19}$$

Using the ridge regression formulation from Eq. 2.7, we can also reversely compute the weights $\boldsymbol{w}_g$ given the trajectory $\boldsymbol{Y}$.

Extending Eq. (3.19) from a single time step to the entire trajectory $\boldsymbol{Y}_{0:T}$, we can now express the trajectory distribution similarly to ProMPs. We consider a system where the current robot state $\boldsymbol{Y}_b, \dot{\boldsymbol{Y}}_b$ can be acquired precisely, which is universal in most robotic systems. In this case, the trajectory's variability is only caused by the variability of the weights $\boldsymbol{w}_g$ plus the observation white noise $\boldsymbol{\epsilon}_y$. Similar to ProMPs, assuming the weights $\boldsymbol{w}_g$ follows a multivariate normal distribution $\boldsymbol{w}_g \sim \mathcal{N}(\boldsymbol{w}_g | \boldsymbol{\mu}_{\boldsymbol{w}_g}, \boldsymbol{\Sigma}_{\boldsymbol{w}_g})$, we can compute the trajectory distribution with full covariance over all time steps $0:T$ and all DoF $1:D$ as

$$p(\boldsymbol{Y}_{0:T}; \boldsymbol{\mu}_{w_g}, \boldsymbol{\Sigma}_{w_g}, \boldsymbol{Y}_b, \dot{\boldsymbol{Y}}_b) = \mathcal{N}(\boldsymbol{Y}_{0:T} | \ \boldsymbol{\mu}_Y, \boldsymbol{\Sigma}_Y), \tag{3.20}$$

$$\begin{aligned} \boldsymbol{\mu}_Y &= \boldsymbol{\xi}_1 \boldsymbol{Y}_b + \boldsymbol{\xi}_2 \dot{\boldsymbol{Y}}_b + \boldsymbol{H}_{0:T}^\top \boldsymbol{\mu}_{w_g}, \\ \boldsymbol{\Sigma}_Y &= \boldsymbol{H}_{0:T}^\top \boldsymbol{\Sigma}_{w_g} \boldsymbol{H}_{0:T} + \sigma_n^2 \mathbf{I}, \end{aligned}$$

with $\boldsymbol{H}_{0:T} = \boldsymbol{\xi}_3 \boldsymbol{\Psi}_b + \boldsymbol{\xi}_4 \dot{\boldsymbol{\Psi}}_b + \boldsymbol{\Psi}_{0:T}$ and $\boldsymbol{\xi}_k = [\xi_k(t)]_{t=0:T}$.

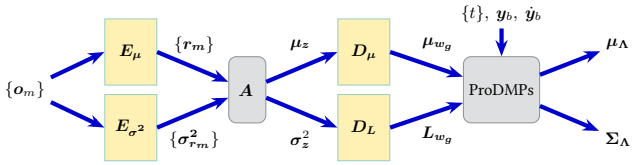


Figure 3.4.: Architecture and pipeline of our deep ProDMPs model. The encoders, aggregator and decoders are denoted by $\boldsymbol{E}$, $\boldsymbol{A}$, and $\boldsymbol{D}$ respectively. Yellow nodes denote deep neural networks, while gray nodes denote operations without learnable parameters.

In a summary, ProDMPs unify DMPs and ProMPs in a consistent framework, inheriting the properties and advantages of both. In Fig. 3.2, we illustrate a few examples to highlight the properties. We show that ProDMPs can generate an identical trajectory as DMPs, and enforce the predicted trajectory distribution adheres to the given initial conditions. The newly generated trajectories are guaranteed to have a smooth transition to the previous trajectory at the replanning time. Besides, ProDMPs inherit all the probabilistic operations of ProMPs, such as combination and blending. For the exact formulations of these operations, we refer to the original ProMPs paper [3], as they remain unaltered. In contrast to previous NN-DMPs methods [33, 22, 15], our model separates the learnable parameters from the numerical integrals which are transformed as position and velocity basis functions. These basis functions are shared by all trajectories to be generated during learning procedure. Hence, we can pre-compute these basis functions once offline at first and use them as constants in online trajectory generation. Consequently, we exclude numerical integrals from forward and backward propagation of NN pipelines, and can thus increase the trajectory generation speed. We present a pipeline comparison between previous methods and our ProDMPs in Fig. 3.3.

3.4. Embed ProDMPs in a deep architecture using Bayesian Set Encoders

We extend ProDMPs with a deep NN architecture that allows for conditioning the trajectory distribution on a set of high-dimensional (time-stamped) sensory observations, such as images. Similar to recent MP architectures [4, 36], we treat such observations as set-inputs [151, 111], i. e. , our architecture is invariant to the order of these observations.

Architecture. The architecture has four major parts: encoder, aggregator, decoder, and a ProDMPs layer, cf. Fig. 3.4. The *Encoders* $E_{\boldsymbol{\mu}}$ and $E_{\boldsymbol{\sigma}^2}$ compute for each observation $\boldsymbol{o}_m \in \mathcal{O}$ a corresponding latent observation $\boldsymbol{r}_m$ and an uncertainty $\boldsymbol{\sigma}^2_{\boldsymbol{r}_m}$, measuring how informative this observation is. The *Bayesian Aggregation* [17] A is a parameter-free operator used to aggregate a set of latent observations $\{\boldsymbol{r}_m\}$ and their uncertainties $\{\boldsymbol{\sigma}^2_{\boldsymbol{r}_m}\}$ into a latent state posterior $p(\boldsymbol{z}|\mathcal{O})$ which is

given by a factorized multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}_{\boldsymbol{z}}, \boldsymbol{\sigma}^2_{\boldsymbol{z}})$. The resulting aggregation is

$$\begin{aligned} \boldsymbol{\sigma}^2_{\boldsymbol{z}|\boldsymbol{o}_{1:M}} &= \left[(\boldsymbol{\sigma}^2_{\boldsymbol{z},0})^{\ominus} + \sum_{m=1}^{M} (\boldsymbol{\sigma}^2_{\boldsymbol{r}_m})^{\ominus} \right]^{\ominus}, \\ \boldsymbol{\mu}_{\boldsymbol{z}|\boldsymbol{o}_{1:M}} &= \boldsymbol{\mu}_{\boldsymbol{z},0} + \boldsymbol{\sigma}^2_{\boldsymbol{z}} \odot \sum_{m=1}^{M} (\boldsymbol{r}_m - \boldsymbol{\mu}_{\boldsymbol{z},0}) \oslash \boldsymbol{\sigma}^2_{\boldsymbol{r}_m}, \end{aligned} \tag{3.21}$$

where $\ominus$, $\odot$, and $\oslash$ denote element-wise inversion, product, and division, respectively. The parameters of the latent variable's prior distribution are $\boldsymbol{\mu}_{\boldsymbol{z},0}$ and $\boldsymbol{\sigma}^2_{\boldsymbol{z},0}$.

Intuitively, BA emphasizes the learning of observation uncertainty $\boldsymbol{\sigma}^2_{\boldsymbol{r}_m}$ and allows to quantify the amount of information contained in an observation. A higher uncertainty will be translated into a little weight of the observation used in the aggregation procedure. Thus, BA can handle task ambiguity more efficiently in the aggregation result than the mean aggregation which recognizes each observation equally important [17]. Hence, we adopt this approach for our architecture. The *Decoders* D_μ, D_L predict the mean $\boldsymbol{\mu}_{\boldsymbol{w}_{\boldsymbol{g}}}$ and the Cholesky decomposition $\boldsymbol{L}_{\boldsymbol{w}_{\boldsymbol{g}}}$ of the covariance $\boldsymbol{\Sigma}_{\boldsymbol{w}_{\boldsymbol{g}}}$ of the weights distribution. Similar to Volpp et al. [17], we do not sample from the latent posterior $p(\boldsymbol{z}|\mathcal{O})$ but use its mean $\boldsymbol{\mu}_z$ and variance $\boldsymbol{\sigma}^2_z$ to predict $\boldsymbol{\mu}_{\boldsymbol{w}_{\boldsymbol{g}}}$ and $\boldsymbol{L}_{\boldsymbol{w}_{\boldsymbol{g}}}$, respectively. Given the query time steps $\{t\}$ and the current robot position $\boldsymbol{Y}_b$ and velocity $\dot{\boldsymbol{Y}}_b$ as initial conditions, the *ProDMPs* layer computes the parameters $\boldsymbol{\mu}_{\boldsymbol{Y}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{Y}}$ of the trajectory distribution as a multivariate Gaussian distribution.

Loss Function. In this work, we focus on Imitation Learning tasks and minimize the negative log-likelihood of the conditional trajectory distribution, i. e. , $-\log \mathcal{N}(\boldsymbol{Y}|\boldsymbol{\mu}_Y, \boldsymbol{\Sigma}_Y)$. Here, $\boldsymbol{Y}$ is the trajectory ground truth, and $\boldsymbol{\mu}_Y, \boldsymbol{\Sigma}_Y$ the mean and covariance of the predicted trajectory distribution conditioned on a set of observations $\mathcal{O}$. Yet, predicting a trajectory's full covariance matrix $\boldsymbol{\Sigma}_Y$ demands high computational resources. The $TD \times TD$ covariance matrix has to be inverted in the loss computation. To keep the computation manageable, we never compute the distribution of the whole time sequence, but only compute the covariance of a pair of time steps, which limits the size of covariance matrices to $2D \times 2D$. We randomly select K such time pairs $\{(t_k, t'_k)\}_{k=1,\dots,K}$, where $t_k, t'_k \in \{0, ..., T\}$, and predict the joint distribution on each of these pairs. As such random selection is executed in every training batch, we can still learn the correlation between time steps while keeping the cost of matrix inversion manageable. Given the boundary conditions $\boldsymbol{Y}_b, \dot{\boldsymbol{Y}}_b$ and a set of observations $\mathcal{O}$, the loss function is thus defined as the mean negative log-likelihood of K random pairs of trajectory values $\boldsymbol{Y}_{(t_k,t'_k)}$ as

$$\mathcal{L}_{\boldsymbol{\theta}}(\boldsymbol{Y}, \boldsymbol{Y}_b, \dot{\boldsymbol{Y}}_b, \mathcal{O}) = -\frac{1}{K} \sum_{k=1}^{K} \log \mathcal{N}(\boldsymbol{Y}_{(t_k,t'_k)}|\boldsymbol{\mu}_{(t_k,t'_k)}, \boldsymbol{\Sigma}_{(t_k,t'_k)}), \tag{3.22}$$

where $\boldsymbol{\mu}_{(t_k,t'_k)}$ and $\boldsymbol{\Sigma}_{(t,t')_k}$ denote the mean and covariance of the joint distribution at the k-th paired time points which are obtained from the ProDMPs decoder depicted in Figure 3.4. In Fig. 3.5, we illustrate that our loss function can capture the temporal correlation. Further, an ablation study

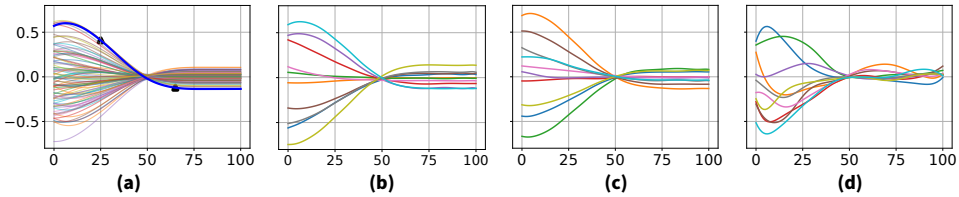


Figure 3.5.: Learning trajectory's temporal correlation using two time steps. x-axis: time steps, y-axis: position [m]. (a) Trajectories in a dataset containing temporal correlation, which can be learned through two randomly selected time steps, i. e., triangle markers. (b) Using full covariance and (c) using two time steps jointly in Eq.(4.4) capture the temporal correlation correctly and reproduce the same result. (d) Maximizing the likelihood of one single time step's value cannot learn the temporal correlation properly.

that only computes the likelihood of single-time steps results in a degenerated estimate of the temporal correlation.

3.5. Experiments

We highlight the advantages of our model and distinguish it from other methods through four experiments, which answer the following questions:

1. Does our model accelerate trajectory computation?
2. Does our model produce high-quality trajectory distributions and samples conditioning on high dimensional observations?
3. Does it support online replanning and predicting a smooth trajectory distribution from the current robot state?
4. Can we condition on several partial observations and obtain a trajectory distribution that leverages the aggregated information?

We compare our method with state-of-the-art CNMPs [4] and NN-based DMPs models [33, 22, 15] on three digit-writing tasks using images as inputs, one simulated robot pushing task with complex physical interaction, and a real robot picking task with shifting object positions.

3.5.1. Trajectory Computation Time Comparison

In TABLE 3.1, we compare the computation time of generating a 2-DoF, 6-second long, 1000-Hz trajectory using the two pipelines shown in Fig. 3.3. We predict a 22-dim $\boldsymbol{w}_g$ from a 3-layer fully connected network (10, 128, 22 neurons resp.). Our model is 200-4600 times faster than the NN-DMPs in different settings, which will translate into a speed-up of 10 times in further full learning experiments.

Table 3.1.: We tested the computation time of both forward and backward pass (**FP**/ **BP**) in a trajectory generation process on an Nvidia® RTX-3080Ti GPU. The keyword **IC** are the settings where the initial conditions are renewed so that the coefficients c_1 and c_2 need to be recomputed. Otherwise, they remain unchanged.

Pipelines	**FP**	**FP + IC**	**BP**	**BP + IC**
NN-DMPs	0.6057 s	0.6145 s	1.5261 s	1.5737 s
ProDMPs	0.00013 s	0.0027 s	0.00105 s	0.0039 s
Speed-up	$\times$ **4659**	$\times$ **227**	$\times$ **1453**	$\times$ **403**

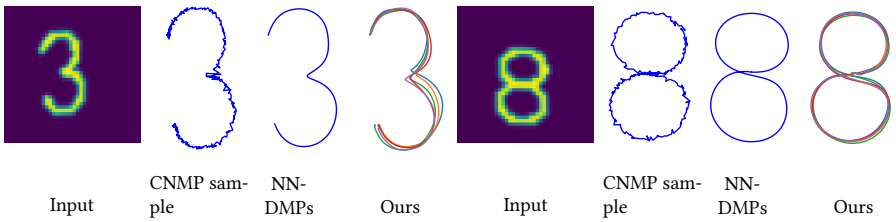


Figure 3.6.: The trajectories generated by different MP models. Only our model captures the temporal and DoF correlations.

3.5.2. Learning Trajectory Distributions of the MNIST Digits

We use a synthetic-MNIST dataset [22] to showcase several properties of our method and advantages over the other approaches. The dataset contains $20,000$ digit images in ten groups (0-9) and as prediction targets 3 seconds trajectories with 2 DoF each. In our settings, we choose to use 25 basis functions per DoF and predict a 52-dim multivariate Gaussian distribution over ProDMPs parameters (25 weights and 1 goal for each DoF). A 2-dim vector for the starting point of the trajectory is also predicted. We analyse our experiments based on three subtasks and show the spatial writing trajectories, i. e. , the x- and y-axis are spatial DoF.

Comparison of Generative Models. As first task, we consider the prediction of trajectories using different methods given one image input. For CNMPs and ProDMPs, we initially predict a distribution and then sample trajectories from it. As shown in Fig. 3.6, CNMPs only model the mean and the isotropic variance per time step and thus fail to model correlations across time steps and dimensions. The NN-DMPs instead predicted a smooth trajectory, but do not learn any statistics over the trajectories and cannot generate samples. Our model, however, not only captures the correct shape of the trajectory but can also be used as a generative model to sample temporal and DoF consistent trajectories. To evaluate the speed of a full learning experiment, we replace the numerical integration of the NN-DMPs with our linear basis function model and keep the network architecture and MSE loss unchanged. The whole training time is reduced from 105 minutes to 10 minutes, which is approximately a speed-up by a factor of 10.

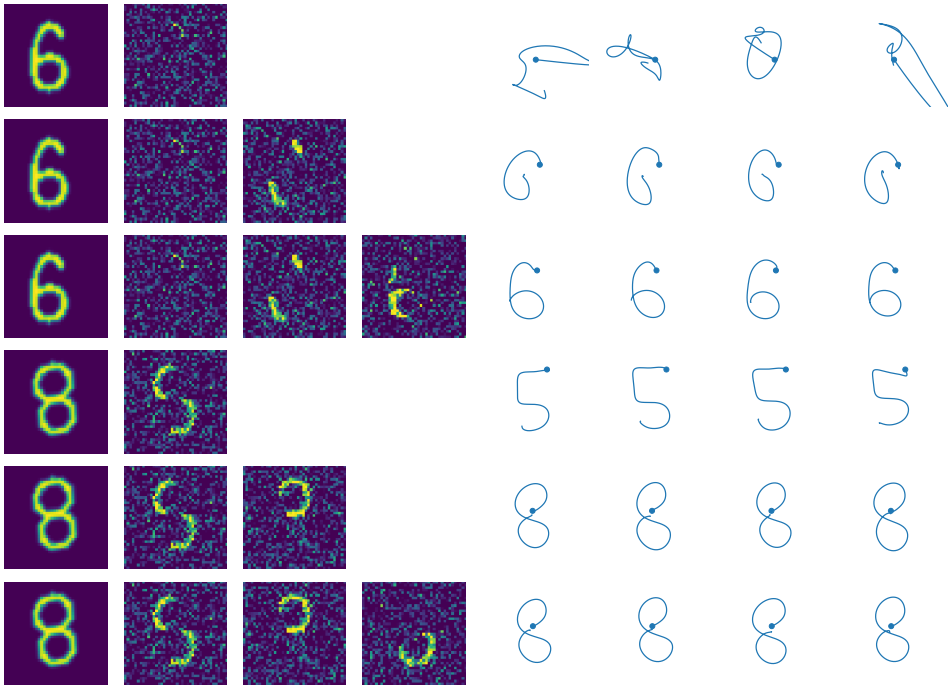


Figure 3.7.: Given up to three noisy images, our model aggregated the info and refined its prediction.

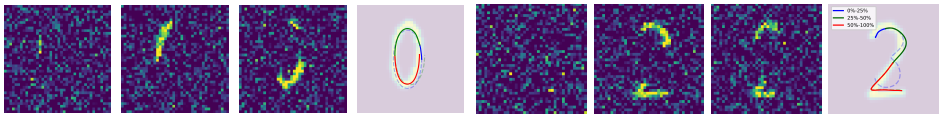


Figure 3.8.: Write a digit in 3 steps. Our model received the first noisy image at the beginning while receiving the second and third at 25% and 50% of the execution time respectively. The trajectory gets replanned (blue → green → red) and executed accordingly. The dashed lines indicate the remaining, not executed trajectories.

Conditioning on Partial Observations. For the second task, we show how our model leverages Bayesian Aggregation to deal with multiple partial observations. We repeatedly add synthetic noise to each original image such that we obtain three noisy images for each digit. The random noise masks occlude most of the image, leaving only a few original pixels visible.

We use our model to progressively make trajectory distribution predictions conditioned on one, two, and three noisy images and sample from each resulting distribution. In Fig. 3.7, we can see our model aggregated the information contained in different observations and provided a refinement progressively (row by row). Without sufficient information, the trajectories appear random or misclassified but become refined once more information is added.

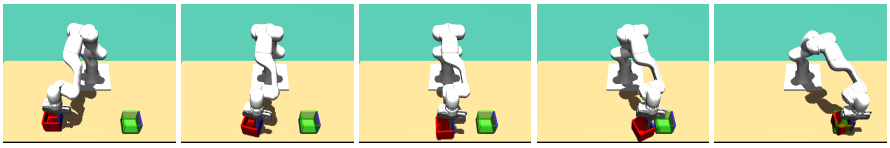


Figure 3.9.: Pushing an empty box (red) from a randomized initial state to a given target position and orientation (green) in Mujoco [152].

Online Replanning. Finally, we show how our method benefits from its dynamical system in online replanning once new information is provided. Different from the previous experiment where the noisy images are given before execution, in this task, we provide new observations to the model during the execution of the trajectory. The same noisy images as in the previous experiment of the target digit are provided at 0%, 25%, and 50% of the execution time. For illustration purposes, we trained the models only on images containing one type of digit or digits with similar spatial and temporal structure. As shown in Fig. 3.8, each additional observation increases the accuracy of the trajectory, while the new trajectories transition smoothly at the replanning points.

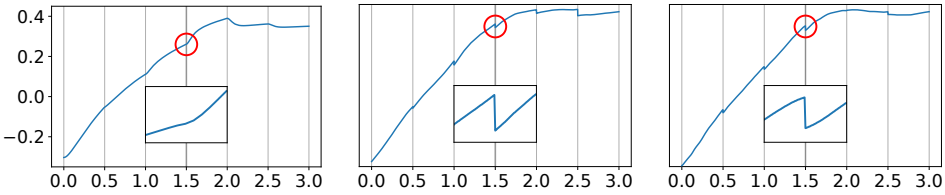
3.5.3. Pushing and Replanning with Complex Contact

We conduct a simulated robot experiment to prove that our approach can deal with rich physical interactions and online conditioning on task variables. We use a Franka Panda robot in Mujoco [152] to push a box with its end-effector, cf. Fig. 3.9. The box has a square shape and is empty. The end-effector is equipped with a peg and starts in the box. Using a simple teleoperation interface, we control the end-effector's x and y position with a mouse, which is then translated into the joint angles using inverse kinematics (IK). We collected 225 demos, each 3 seconds long, moving the box from a random initial state to a desired target location and orientation. The box is hard to control, especially the orientation, due to the non-linear interaction of its walls and corners with the peg. The learning procedure is as follows. At each iteration, we randomly choose a replanning time and use the recorded box position, orientation, as well as the actual robot end-effector's position and velocity in the past 0.1s as observations. We predict a desired end-effector trajectory distribution in the next 0.5s using ProDMPs with 25 basis functions per DoF and maximize the log-likelihood of the demonstrated trajectory. In inference, our model directly interacts with the simulator, pushing a box from an unseen initial state. We executed either the mean or the sampled trajectory from the prediction, and compare our method with

- CNMPs using the mean and sampling,
- ProMPs using the mean and sampling (replace the ProDMP layer by a ProMPs layer in Fig. 3.4),
- ProDMPs without replanning, and
- A vanilla behavior cloning network mapping observations to actions at each time step.

Table 3.2.: Evaluation of different settings using (a) Mean and std of the absolute error (x-/y-position and orientation along z-axis) of the final box state to its target state, as well as (b) the average squared acceleration (ASA) as smoothness metric. Each model is trained 20 times using different random seeds.

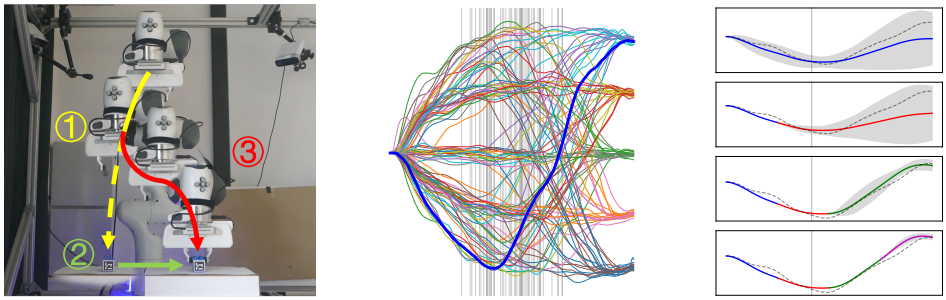
	Replan	x (mm)	y (mm)	z-ori. (deg)	ASA
Demos	-	4.2 ± 2.9	4.8 ± 3.4	1.76 ± 1.28	-
ProDMP	mean	$\mathbf{9.2 \pm 8.2}$	$\mathbf{7.1 \pm 7.2}$	$\mathbf{4.3 \pm 5.7}$	3.3
CNMP	mean	9.5 ± 9.2	7.5 ± 9.7	4.7 ± 9.0	800.5
ProMP	mean	9.5 ± 8.6	7.6 ± 7.1	4.8 ± 6.0	682.9
ProDMP	sample	19.0 ± 31.3	17.6 ± 20.1	12.1 ± 21.1	13.1
CNMP	sample	72.4 ± 13.7	80.8 ± 21.5	20.8 ± 14.4	2.7e4
ProMP	sample	24.4 ± 16.3	25.8 ± 14.7	11.1 ± 14.4	2379
ProDMP	$\times$	62.0 ± 53.9	55.9 ± 38.8	33.6 ± 28.4	**0.3**
B. C.	per step	35.8 ± 34.0	94.9 ± 14.2	21.2 ± 24.0	7.1



(a) ProDMPs' mean **(b)** CNMPs' mean **(c)** ProMPs' mean

Figure 3.10.: Replanning trajectories comparison of robot pushing. x-axis: time [s], y-axis: end-effector y position [m]. Each trajectory is a concatenation of 6 replanning segments, 0.5 second long each. Our model is smooth at the replanning time steps (vertical lines), while the CNMPs and ProMPs are discontinuous when replanning.

We evaluate the error between the final and target states of the box, as well as the average squared acceleration (ASA, the smaller the better) as a trajectory smoothness metric. The result is shown in Table 3.2. Our model can reproduce the box-pushing task with similar accuracy as the demos and achieves excellent levels of smoothness. The performance of the CNMPs and ProMPs' mean prediction yields a similar pushing result to our model. However, the trajectory samples of CNMPs are, as expected, unable to solve the task. In addition, CNMPs and ProMPs exhibit noticeable jumps and significant accelerations at the transition points of the trajectory. The sampled trajectories of CNMPs are lacking temporal and DoF correlation, leading to an extremely low level of smoothness. The ablated case of our model cannot solve the task properly, indicating that replanning is crucial to adjust the movement in such a contact-rich scenario. The comparably high smoothness is achieved by the lack of replanning in the trajectory. The state-action-based behavioral cloning using an MSE



(a) Real Setup **(b)** Collected Demonstrations **(c)** The 3rd joint position

Figure 3.11.: (a) The robot is picking an object initialized on the left. During the movement, the object was shifted to the right, and the robot replanned a new trajectory from its current state to the new object position without interruption. (b) For each episode, every 1 second our model predicts a new distribution using its current state and the latest box position. Before the shift (vertical line), the predictions contain high variance. After the shift, the variance decreases intensely. The dashed line denotes the human's demo.

loss cannot capture the temporal correlation between the box and the robot's movement, and thus performs poorly. It is worth mentioning that using NN-based DMPs in the current task is equivalent to training our model with the MSE loss, which will lead to the same result but a faster speed.

3.5.4. Object Picking with Dynamic Positional Shift

Finally, we learn an object-picking task in a 7-DoF joint space. A Franka Panda robot picks an object in its workspace (cf. Fig. 3.11a). During execution, the object shifts to a new position, and the robot must plan a new trajectory from its current position to pick up the object without interruption. We collected 100 human demos, each 4 seconds long, using the same teleoperation interface as in Section 3.5.3. In each demo, the human moves the robot end-effector to the object and adjusts the movement when the object shifts. The demonstrated joint trajectory is computed using IK. After each second, the actual object position and the robot state are given to ProDMPs to replan a trajectory distribution starting from the current robot state. We evaluated the picking success rate of our model using the

(a) mean prediction, and

(b) trajectory samples in a simulated and a real setup.

The box position in the real setup is captured by a camera system using ArUco ROS [153]. Our model learns picking and replanning movement directly from the data and achieves a high success rate in simulation, as shown in TABLE 3.3. Due to the camera delay in the real setup, the performance decreases slightly. Fig. 3.11c shows four predicted trajectory distributions in one sequence. Our model predicts the trajectory distribution with a high variance at the beginning and decreases the predicted variance once it observes the shift of the object.

Table 3.3.: Success rate and shift distance of 100 picks

100 picks of	Sim+mean	Sim+sample	Real+mean	Real+sample
Success rate	99%	82%	87%	75%
Shift distance	22.8 ± 5.6		21.7 ± 9.7	21.0 ± 8.0

3.6. Conclusion

We presented ProDMPs, a unified framework fusing dynamic and probabilistic movement primitives. ProDMPs recovered a linear basis-function representation for the trajectories by solving the ODE of the dynamical system. This way, we can easily represent trajectory distributions that adhere to initial conditions defined by the current robot position and velocity as well as generate smooth trajectories when replanning. Further, we built a neural aggregation model for non-linear iterative conditioning and found a solution to learn full trajectory covariances with fewer resources. Our deep embedded ProDMPs achieved smoothness, goal convergence, trajectory correlation modeling, non-linear trajectory conditioning, and online replanning in one model. For future work, we will extend our approach to reinforcement learning, and consider force profiles that need to be applied. We expect our model can process data with multi-sensor modalities and thus achieve sensor fusion in the latent space. If the data has strong temporal order and causality, using a recurrent structure or a transformer may increase the performance. For robotic experiments, using raw camera images as input will make the tasks more challenging.

4. Temporally Correlated Episodic Reinforcement Learning

Traditional Episodic Reinforcement Learning (ERL) employs MPs as trajectory generators and treats each trajectory as an indivisible unit [10]. This practice discards rich step-wise state–action information, yielding slow and sample-inefficient policy updates despite ERL's broad exploration and smooth motion generation. Hence **[RQ2]** asks: *How can fine-grained step-based information be integrated into episodic RL to enhance learning efficiency?*

Contributions. I answer this question with *Temporally-Correlated Episodic RL (TCE)*, which injects detailed step-level feedback into episodic policy optimisation. TCE partitions each episode into short segments and estimates the contribution of every segment to the overall return, supplying informative credits for policy updates that replace the coarse trajectory-level signals of standard ERL.

TCE is grounded in the probabilistic ProDMP framework [12], allowing exact segment-wise likelihood evaluation. By exploiting intermediate rewards and states, it markedly improves data efficiency and lowers the sample complexity traditionally associated with episodic training.

Extensive experiments on manipulation tasks with both dense and sparse rewards show that TCE outperforms baseline ERL and step-based RL. It fuses the fine-grained credit assignment of step-based methods with the smooth, temporally consistent trajectories of ERL, thereby broadening the practical utility of episodic RL in robotics.

The remainder of this chapter has been published as [13]: Ge Li, Hongyi Zhou, Dominik Roth, Serge Thilges, Fabian Otto, Rudolf Lioutikov, and Gerhard Neumann. "Open the Black Box: Step-based Policy Updates for Temporally-Correlated Episodic Reinforcement Learning." *In: International Conference on Learning Representation (ICLR)*, 2024.

4.1. Introduction

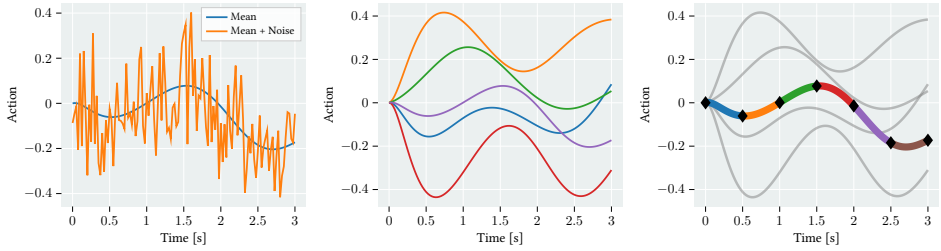
By considering how policies interact with the environment, reinforcement learning (RL) methodologies can be classified into two distinct categories: step-based RL (SRL) and episodic RL (ERL). SRL predicts actions for each perceived state, while ERL selects an entire behavioral sequence at the start of an episode. Most predominant deep RL methods, such as PPO [7] and SAC [8], fall into the category of SRL. In these methods, the step information — comprising state, action, reward,

subsequent state, and done signal received by the RL agent at each discrete time step — is pivotal for policy updates. This granular data aids in estimating the policy gradient [46, 47], approximating state or state-action value functions [8], and assessing advantages [49]. Although SRL methods have achieved great success in various domains, they often face significant exploration challenges. Exploration in SRL, often based on a stochastic policy like a factorized Gaussian, typically lacks temporal and cross-DoF (degrees of freedom) correlations. This deficiency leads to inconsistent and inefficient exploration across state and action spaces [154, 155], as shown in Fig. 4.1a. Furthermore, the high variance in trajectories generated through such exploration can cause suboptimal convergence and training instability, a phenomenon highlighted by considerable performance differences across various random seeds [156].

Episodic RL. in contrast to SRL, represents a distinct branch of RL that emphasizes the maximization of returns over entire episodes [9, 90, 91], rather than focusing on the internal evolution of the environment interaction. This approach shifts the solution search from per-step actions to a parameterized trajectory space, employing techniques like Movement Primitives (MPs) [1, 3]. Such exploration strategy allows for broader exploration horizons and ensures consistent trajectory smoothness across task episodes, as illustrated in Fig. 4.1b. Additionally, it is theoretically capable of capturing temporal correlations and interdependencies among DoF. ERL typically treats entire trajectories as single data points, often overlooking the internal changes in the environment and state transitions. This approach leads to training predominantly using black-box optimization methods [10, 93, 98, 102]. The term *black box* in our title reflects this reliance on black-box optimization, which tends to overlook detailed step-based information acquired during environmental interactions. However, this often results in a lack of attention to the individual contributions of each segment of the trajectory to the overall task success. Consequently, while ERL excels in expansive exploration and maintaining trajectory smoothness, it typically requires a larger volume of samples for effective policy training. In contrast, step-based RL methods have demonstrated notable advancements in learning efficiency by utilizing this detailed step-based information.

Open the Black Box. In this work, our goal is to integrate step-based information into the policy update process of ERL. Our proposed method, Temporally-Correlated Episodic RL (TCE), moves beyond the traditional approach of treating an entire trajectory as a single data point. Instead, we transform trajectory-wide elements, such as reproducing likelihood and advantage, into their segment-wise counterparts. This enables us to leverage the step-based information to recognize and accentuate the unique contributions of each trajectory segment to overall task success. Through this innovative approach, we have opened the black box of ERL, making it more effective while retaining its strength. As a further step, we explore the benefits of fully-correlated trajectory exploration in deep ERL. We demonstrate that leveraging full covariance matrices for trajectory distributions significantly improves policy quality in existing black-box ERL methods like [10].

Our contributions are summarized as:



(a) Step-based RL **(b)** Traj.-based, Episodic RL **(c)** TCE (ours)

Figure 4.1.: Illustration of exploration strategies: (a) SRL samples actions by adding noise to the predicted mean, resulting in inconsistent exploration and jerky actions. However, their leverage of step-based information leads to efficient policy updates. (b) ERL samples complete trajectories in a parameter space and generate consistent control signals. Yet, they often treat trajectories as single data points and overlook the step-based information during the interaction, causing inefficient policy update. (c) TCE combines the benefits of both, using per-step information for policy update while sampling complete trajectories with broader exploration and high smoothness.

1. We propose TCE, a novel RL framework that integrates step-based information into the policy updates of ERL, while preserving the broad exploration scope and trajectory smoothness characteristic of ERL.
2. We provide an in-depth analysis of exploration strategies that effectively capture both temporal and degrees of freedom (DoF) correlations, demonstrating their beneficial impact on policy quality and trajectory smoothness.
3. We conduct a comprehensive evaluation of our approach on multiple simulated robotic manipulation tasks, comparing its performance against other baseline methods.

4.2. Related Work

Improve Exploration and Smoothness in Step-based RL. SRL methods, such as PPO and SAC, interact with the environment by performing a sampled action at each time-step. This strategy often results in a control signal with high-frequency noise, making it unsuitable for direct use in robotic systems [154]. A prevalent solution is to reduce the sampling frequency, a technique commonly known as *frame skip* [157]. Here, the agent only samples actions every k-th time step and replicates this action for the skipped steps. Similar approaches decide whether to repeat the last action or to sample a new action in every time step [158, 159]. This concept is also echoed in works such as general State Dependent Exploration (gSDE) [154, 160, 161], where the applied noise is sampled in a state-dependent fashion; leading to smooth changes of the disturbance between consecutive steps.

However, while these methods improved the smoothness in small segments, they struggled to model long-horizon correlations. Another area of concern is the utilization of white noise during sampling,

which fails to consider the temporal correlations between time steps and results in a random walk with suboptimal exploration. To mitigate this, previous research, such as [88] and [162], have integrated colored noise into the RL policy, aiming to foster exploration that is correlated across time steps. While these methods have shown advantages over white noise approaches, they neither improve the trajectory's smoothness, nor adequately capture the cross-DoF correlations.

Episodic RL. The early ERL approaches used black-box optimization to evolve parameterized policies, e.g., small NN [90, 91, 92]. However, these early works were limited to tasks with low-dimensional action space, for instance, the Cart Pole. Although recent works [93, 94] have shown that, with more computing, these methods can achieve comparable asymptotic performance with step-based algorithms in some locomotion tasks, none of these methods can deal with tasks with context variations (e.g., changing goals). Another research line in ERL works with more compact policy representation. [9, 163] first combined ERL with MPs, reducing the dimension of searching space from NN parameter space to MPs parameter space with the extra benefits of smooth trajectories generation. [62] proposed a model-based method to improve the sample efficiency. Notably, although those methods can already handle some complex manipulation tasks such as robot baseball [9], none of them can deal with contextual tasks. To deal with that problem, [97] further extends that utilizes linear policy conditioned on the context. Another recent work from this research line [98] proposed using a Mixture of Experts (MoE) to learn versatile skills under the ERL framework.

BBRL. As the first method that utilizes non-linear adaptation to contextual ERL, Deep Black Box Reinforcement Learning (BBRL) [10] is the most related work to our method. BBRL applies trust-region-constrained policy optimization to learn a weight adaptation policy for MPs. Despite demonstrating great success in learning tasks with sparse and non-Markovian rewards, it requires significantly more samples to converge compared to SoTA SRL methods. This could be attributed to its black-box nature, where the trajectory from the entire episode is treated as a single data point, and the trajectory return is calculated by summing up all step rewards within the episode.

4.3. Preliminaries

4.3.1. Episodic Reinforcement Learning

Markov Decision Process (MDP). We consider a MDP problem of a policy search defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \mathcal{P}_0, \gamma)$. We assume the state space $\mathcal{S}$ and action space $\mathcal{A}$ are continuous and the transition probabilities $\mathcal{T} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ describe the state transition probability to $\boldsymbol{s}_{t+1}$, given the current state $\boldsymbol{s}_t \in \mathcal{S}$ and action $\boldsymbol{a}_t \in \mathcal{A}$. The initial state distribution is denoted as $\mathcal{P}_0 : \mathcal{S} \rightarrow [0, 1]$. The reward $r_t(\boldsymbol{s}_t, \boldsymbol{a}_t)$ returned by the environment is given by a function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ and $\gamma \in [0, 1]$ describes the discount factor. The goal of RL in general is to find a policy π that maximizes the expected accumulated reward, namely return, as $R = \mathbb{E}_{\mathcal{T}, \mathcal{P}_0, \pi}[\sum_{t=0}^{\infty} \gamma^t r_t]$.

Episodic RL [90] focuses on maximizing the return $R = \sum_{t=0}^{T} [\gamma^t r_t]$ over a task episode of length T, irrespective of the state transitions within the episode. This approach typically employs

a parameterized trajectory generator, like MPs [1], to predict a trajectory parameter vector $\boldsymbol{w}$. This vector is then used to generate a complete reference trajectory $\boldsymbol{a}(\boldsymbol{w}) = \boldsymbol{a}_{0:T}$. The resulting trajectory is executed using a trajectory tracking controller to accomplish the task. In this context, $\boldsymbol{a}_t \in \mathbb{R}^D$ denotes the trajectory value at time t for a system with D DoF, differentiating it from the per-step action $\boldsymbol{a}$ used in SRL. In contrast to Chapter 3, the trajectories generated by MPs are directly used as actions for RL. Therefore, we use $\boldsymbol{a}$ instead of $\boldsymbol{a}$ to align with standard RL notation. It is important to note that, although ERL predicts an entire action trajectory, it still maintains the *Markov Property*, where the state transition probability depends only on the given current state and action [45]. In this respect, the action selection process in ERL is fundamentally similar to techniques like action repeat [157] and temporally correlated action selection [154, 162]. In contrast to SRL, ERL predicts the trajectory parameters as $\pi(\boldsymbol{w}|\boldsymbol{s})$, which shifts the solution search from the per-step action space $\mathcal{A}$ to the parameter space $\mathcal{W}$. Therefore, a trajectory parameterized by a vector $\boldsymbol{w}$ is typically treated as a single data point in $\mathcal{W}$. Consequently, ERL commonly employs black-box optimization methods for problem-solving [93, 10]. The general learning objective of ERL is formally expressed as

$$J = \int \pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{s})[R(\boldsymbol{s}, \boldsymbol{w}) - V^{\pi}(\boldsymbol{s})]d\boldsymbol{w} = \mathbb{E}_{\boldsymbol{w}\sim\pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{s})}[A(\boldsymbol{s}, \boldsymbol{w})], \tag{4.1}$$

where $\pi_{\boldsymbol{\theta}}$ represents the policy, parameterized by $\boldsymbol{\theta}$, e. g. using NNs. The initial state $\boldsymbol{s} \in \mathcal{S}$ characterizes the starting configuration of the environment and the task goal, serving as the input to the policy. The $\pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{s})$ indicates the likelihood of selecting the trajectory parameter $\boldsymbol{w}$. The term $R(\boldsymbol{s}, \boldsymbol{w}) = \sum_{t=0}^{T}[\gamma^t r_t]$ represents the return obtained from executing the trajectory, while $V^{\pi}(\boldsymbol{s}) = \mathbb{E}_{\boldsymbol{w}\sim\pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{s})}[R(\boldsymbol{s}, \boldsymbol{w})]$ denotes the expected return across all possible trajectories under policy $\pi_{\boldsymbol{\theta}}$. Their subtraction is defined as the advantage function $A(\boldsymbol{s}, \boldsymbol{w})$, which quantifies the benefit of selecting a specific trajectory. By using parameterized trajectory generators like MPs, ERL benefits from consistent exploration, smooth trajectories, and robustness against local optima, as noted by [10]. However, its policy update strategy incurs a trade-off in terms of learning efficiency, as valuable step-based information is overlooked during policy updates. Furthermore, existing method like [15, 10] commonly use factorized Gaussian policies, which inherently limits their capacity to capture all relevant movement correlations.

4.3.2. Using Movement Primitives for Trajectory Representation

The Movement Primitives (MP), as a parameterized trajectory generator, play an important role in ERL and robot learning. This section highlights key MP methodologies and their mathematical foundations, as previously discussed in Chapter 3. [1] introduced the Dynamic Movement Primitives (DMPs) method, incorporating a force signal into a dynamical system to produce smooth trajectories from given initial robot states. Following this, [3] developed Probabilistic Movement Primitives (ProMPs), which leverages a linear basis function representation to map parameter vectors to trajectories and their corresponding distributions. The probability of observing a trajectory $a_{0:T}$

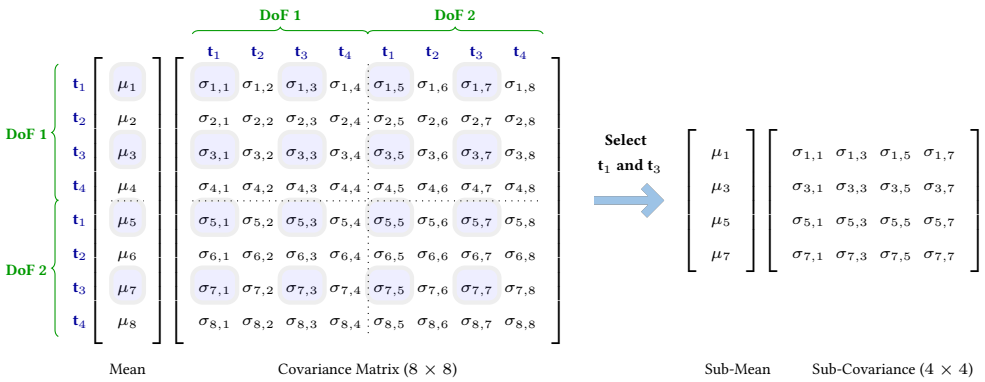


Figure 4.2.: Reduce the trajectory distribution dimensions using two time steps [12], shown in an element-wise format. Here, the trajectory has two DoF and four time steps, with $D \cdot T = 8$. **Left**: The 8-dim mean vector and the 8×8-dim covariance matrix of the original trajectory distribution, capture correlations across both DoF and time steps. **Right**: Randomly selecting two time points, e. g. t_1 and t_3, yields a reduced distribution while still capturing the movement correlations.

given a specific weight vector distribution $p(\boldsymbol{w}) \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu_w}, \boldsymbol{\Sigma_w})$ is represented as a linear basis function model:

$$\boldsymbol{a}_{0:T} = \boldsymbol{\Psi}_{0:T}^{\intercal}\boldsymbol{w}, \tag{4.2}$$

$$p(\boldsymbol{a}_{0:T};\ \boldsymbol{\mu_a}, \boldsymbol{\Sigma_a}) = \mathcal{N}(\boldsymbol{\Psi}_{0:T}^{\intercal}\boldsymbol{\mu_w},\ \boldsymbol{\Psi}_{0:T}^{\intercal}\boldsymbol{\Sigma_w}\boldsymbol{\Psi}_{0:T}\ + \sigma_a^2\boldsymbol{I}). \tag{4.3}$$

Here, variance σ_a^2 is a white noise term, played as a regularizer. The matrix $\boldsymbol{\Psi}_{0:T}$ houses the basis functions for each time step t. Additionally, $p(\boldsymbol{a}_{0:T};\ \boldsymbol{\mu_a}, \boldsymbol{\Sigma_a})$ defines the trajectory distribution coupling the DoF and time steps, mapped from $p(\boldsymbol{w})$.

Chapter 3 introduced Probabilistic Dynamic Movement Primitives (ProDMPs), a hybrid approach that blends the pros of both DMPs and ProMPs. Therefore, we use ProDMPs as the MP model in the current chapter.

4.3.3. Representation of Trajectory Distribution and Likelihood

Computing the trajectory distribution and reconstruction likelihood is crucial for policy updates in ERL. Previous methods like [10, 15] represented the trajectory distribution using the parameter distribution $p(\boldsymbol{w})$ and the likelihood of a sampled trajectory $\boldsymbol{a}^*$ with its parameter vector as $p(\boldsymbol{w}^*|\boldsymbol{\mu_w}, \sigma_{\boldsymbol{w}}^2)$. However, this approach treats an entire trajectory as a singular data point and fails to efficiently utilize step-based information. In contrast, research in imitation learning, including works by [3, 6], maps parameter distributions to trajectory space and allows the exploitation of trajectory-specific information. Yet, the likelihood computation in this space is computationally intensive, primarily due to the need to invert a high-dimensional covariance matrix, a process with an $O((D \cdot T)^3)$ time complexity. Recent studies, like those by [4, 36, 37], advocates for directly

modeling the trajectory distribution using neural networks. These methods typically employ a factorized Gaussian distribution $\mathcal{N}(\boldsymbol{a}|\mu_{\boldsymbol{a}}, \sigma^2_{\boldsymbol{a}})$, instead of a full Gaussian distribution $\mathcal{N}(\boldsymbol{a}|\boldsymbol{\mu}_{\boldsymbol{a}}, \boldsymbol{\Sigma}_{\boldsymbol{a}})$ that accounts for both the DoF and time steps. This choice mitigates the computational burden of likelihood calculations, but comes at the cost of neglecting key temporal correlations and interactions between different DoF. To address these challenges, [12] introduced a novel approach for estimating the trajectory likelihood with a set of paired time points $(t_k, t'_k), k = 1, ..., K$, as

$$\log p(\boldsymbol{a}_{0:T}) \approx \frac{1}{K} \sum_{k=1}^{K} \log \mathcal{N}(\boldsymbol{a}_{(t_k,t'_k)}|\boldsymbol{\mu}_{(t_k,t'_k)}, \boldsymbol{\Sigma}_{(t_k,t'_k)}), \tag{4.4}$$

As shown in Fig. 4.2, this method scales down the dimensions of a trajectory distribution from $D \cdot T$ to a more manageable $D \cdot 2$. Through the use of batched, randomly selected time pairs during training, the method is proved to efficiently capture correlations while reducing computational cost.

4.3.4. Using Trust Regions for stable policy update

In ERL, the parameter space $\mathcal{W}$ typically exhibits higher dimensionality compared to the action space $\mathcal{A}$. This complexity presents unique challenges in maintaining stable policy updates. Trust regions methods [55, 7] has long been recognized as an effective technique for ensuring the stability and convergence of policy gradient methods. While popular methods such as PPO approximate trust regions using surrogate cost functions, they lack the capacity for exact enforcement. To tackle this issue, [59] introduced trust region projection layer (TRPL), a mathematically rigorous and scalable technique that precisely enforces trust regions in deep RL algorithms, as previously discussed in Chapter 2.2.4. By incorporating differentiable convex optimization layers [164], this method not only allows for trust region enforcement for each input state, but also demonstrates significant effectiveness and stability in high-dim parameter space, as validated in method like BBRL [10]. The TRPL takes standard outputs of a Gaussian policy—namely, the mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ —and applies a state-specific projection operation to maintain trust regions. The adjusted Gaussian policy, parameterized by $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\Sigma}}$, forms the basis for subsequent computations. Let d_{mean} and d_{cov} be the dissimilarity measures, e. g. KL-divergence, for mean and covariance, bounded by ϵ_μ and ϵ_Σ respectively. The optimization for each state $\boldsymbol{s}$ is formulated as:

$$\begin{aligned} &\arg\min_{\tilde{\boldsymbol{\mu}}_s} d_{\text{mean}}\left(\tilde{\boldsymbol{\mu}}_s, \boldsymbol{\mu}(\boldsymbol{s})\right), \quad \text{s.t.} \quad d_{\text{mean}}\left(\tilde{\boldsymbol{\mu}}_s, \boldsymbol{\mu}_{\text{old}}(\boldsymbol{s})\right) \leq \epsilon_\mu, \text{ and} \\ &\arg\min_{\tilde{\boldsymbol{\Sigma}}_s} d_{\text{cov}}\left(\tilde{\boldsymbol{\Sigma}}_s, \boldsymbol{\Sigma}(\boldsymbol{s})\right), \quad \text{s.t.} \quad d_{\text{cov}}\left(\tilde{\boldsymbol{\Sigma}}_s, \boldsymbol{\Sigma}_{\text{old}}(\boldsymbol{s})\right) \leq \epsilon_\Sigma. \end{aligned} \tag{4.5}$$

4.4. Use Step-based Information for ERL Policy Updates

We introduce an innovative framework of ERL that builds on traditional ERL foundations, aiming to facilitate an efficient policy update mechanism while preserving the intrinsic benefits of ERL. The

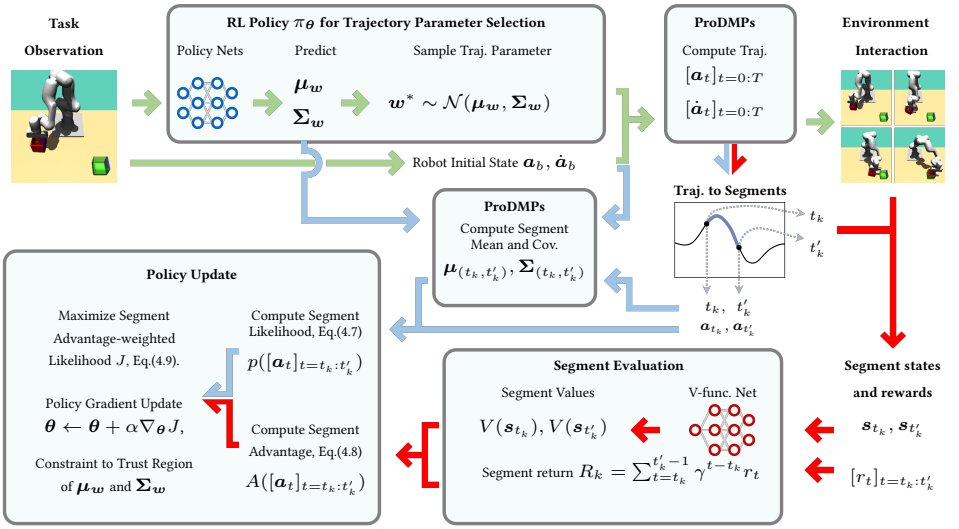


Figure 4.3.: The TCE framework. The entire learning framework can be divided into three main parts. The first part, shown in green arrows, involves trajectory sampling, generation, and execution, detailing how the robot is controlled to complete a given task. The second part, indicated in blue arrows, focuses on estimating the likelihood of selecting a particular segment of the sampled trajectory. The third part, marked by red arrows, deals with segment evaluation and advantage computation, assessing how much each segment contributes to the successful task completion.

key innovation lies in redefining the role of trajectories in the policy update process. In contrast to previous methods which consider an entire trajectory as a single data point, our approach breaks down the trajectory into individual segments. Each segment is evaluated and weighted based on its distinct contribution to the task success. This method allows for a more effective use of step-based information in ERL. The comprehensive structure of this framework is depicted in Figure 4.3.

Trajectory Prediction and Generation. As highlighted by green arrows in figure 4.3, we adopt a structure similar to previous ERL works, such as the one described by [10]. However, this part distinguish itself by using the most recent ProDMPs for trajectory generation and distribution modeling, due to the improved support for trajectory initialization. Additionally, we enhance the previous framework by using a full covariance matrix policy $\pi(\boldsymbol{w}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}})$ as opposed to a factorized Gaussian policy, to capture a broader range of movement correlations.

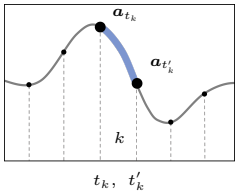


Figure 4.4.: Divide a trajectory into K segments.

Trajectory Likelihood Representation. In RL, the likelihood of previously sampled actions, along with their associated returns, is often used to adjust the chance of these actions being selected in future policies. In previous ERL methods, this process typically involves the probability of choosing an entire trajectory. However, our framework adopts a different strategy, as shown in blue arrows in figure 4.3. Using the techniques in Sections 4.3.2 and 4.3.3, our approach begins by selecting K

paired time steps. We then transform the parameter likelihood into a trajectory likelihood, which is calculated using these K pairwise likelihoods. This approach, depicted in Figure 4.4, effectively divides the whole trajectory into K distinct segments, with each segment defined by a pair of time steps. In essence, this method allows us to break down the overall trajectory likelihood into individual segment likelihoods, offering a more detailed view of the trajectory's contribution to task success.

$$\text{Trajectory to Segments:} \qquad p(\boldsymbol{a}_{0:T}|\boldsymbol{s}) \triangleq \{p(\boldsymbol{a}_{t_k:t'_k}|\boldsymbol{s})\}_{k=1\dots K}, \tag{4.6}$$

$$\text{Local Representation:} \qquad p(\boldsymbol{a}_{t_k:t'_k}|\boldsymbol{s}) \triangleq p(\boldsymbol{a}_{t_k}, \boldsymbol{a}_{t'_k}|\boldsymbol{\mu}_{(t_k,t'_k)}(\boldsymbol{s}), \boldsymbol{\Sigma}_{(t_k,t'_k)}(\boldsymbol{s})). \tag{4.7}$$

Definition of Segment Advantages. Similar to standard SRL methods, we leverage the advantage function to evaluate the benefits of executing individual segments within a sampled trajectory. When being at state $\boldsymbol{s}_{t_k}$ and following the trajectory segment $[\boldsymbol{a}_t]_{t=t_k:t'_k}$, the segment-wise advantage function $A(\boldsymbol{s}_{t_k}, [\boldsymbol{a}_t]_{t=t_k:t'_k})$ quantifies the difference between the actual return obtained by executing this sampled trajectory segment and the expected return from a randomly chosen segment, as

$$A(\boldsymbol{s}_{t_k}, [\boldsymbol{a}_t]_{t=t_k:t'_k}) = \sum_{t=t_k}^{t=t'_k-1} \gamma^{t-t_k} r_t + \gamma^{t'_k-t_k} V^{\pi_{\text{old}}}(\boldsymbol{s}_{t'_k}) - V^{\pi_{\text{old}}}(\boldsymbol{s}_{t_k}), \tag{4.8}$$

where $V^{\pi_{\text{old}}}(\boldsymbol{s}_{t_k})$ denotes the value function of the current policy. In our method, the estimation of $V^{\pi_{\text{old}}}(\boldsymbol{s}_{t_k})$ is consistent with the approaches commonly used in SRL and is independent of the design choice of segment selections. We use NNs to estimate $V^{\pi}(\boldsymbol{s}) \approx V^{\pi}_{\phi}(\boldsymbol{s})$ which is fitted on targets of true return or obtained by general advantage estimation (GAE) [49].

Learning Objective. By replacing the trajectory likelihood and advantage with their segment-based counterparts in the original ERL learning objective as stated in equation 5.5, we propose the learning objective of our method as follows

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\text{old}}}\left[\frac{1}{K}\sum_{k=1}^{K} \frac{p_{\pi_{\text{new}}}(\boldsymbol{a}_{t_k:t'_k}|\boldsymbol{s})}{p_{\pi_{\text{old}}}(\boldsymbol{a}_{t_k:t'_k}|\boldsymbol{s})} A^{\pi_{\text{old}}}(\boldsymbol{s}_{t_k}, \boldsymbol{a}_{t_k:t'_k})\right]. \tag{4.9}$$

Here, $\boldsymbol{s}$ denotes the initial state of the episode, used for selecting the parameter vector $\boldsymbol{w}$, and $\boldsymbol{s}_{t_k}$ is the state of the system at time t_k. The learning objective takes the *Importance Sampling* to update policies using data from previous policies [55, 7, 59]. Our method retains the advantages of exploration in parameter space and generating smooth trajectories. This enables us to enhance the likelihood of segments with high advantage and reduce the likelihood of less rewarding ones during policy updates. To ensure a stable update for the full covariance Gaussian policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}})$, we deploy a differentiable Trust Region Projection step [59] after each policy update iteration as previously discussed in Section 4.3.4.

Summary Algorithm 3 summarizes the method of TCE.

Algorithm 3 Temporally-Correlated Episodic RL (TCE)

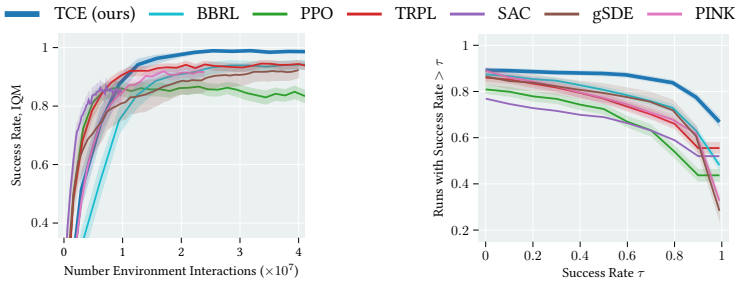
Initialize policy parameters θ and value function parameters ϕ
for iteration = 1, 2, ... **do**
  Get the initial state $\boldsymbol{s}_0$
  Predict the mean $\boldsymbol{\mu}_{\boldsymbol{w}}$, covariance $\boldsymbol{\Sigma}_{\boldsymbol{w}}$, and sample $\boldsymbol{w}*$
  Generate the trajectory $\boldsymbol{y}*$ using Eq.(4.2) and execute it in the environment
  Collect step-based information through the execution
  Update ϕ, use true return or GAE style return [49]
  Select K time-pairs, e.g. choose every 10 steps along the trajectory
  Compute the segment-wise likelihood $\{p_k^{\text{old}}\}_{k=1:K}$ using Eq.(4.6) and 4.7 under π^{old}
  for update epoch = 1, 2, ... **do**
    Make prediction of the mean $\boldsymbol{\mu}_{\boldsymbol{w}}^{\text{new}}$, covariance $\boldsymbol{\Sigma}_{\boldsymbol{w}}^{\text{new}}$ under the latest policy π^{new}
    Enforce Trust Region by projecting $\boldsymbol{\mu}_{\boldsymbol{w}}^{\text{new}}$ and $\boldsymbol{\Sigma}_{\boldsymbol{w}}^{\text{new}}$ through TRPL using Eq.(4.5)
    Get projected policy $\tilde{\pi}^{\text{new}}$, represented by $\tilde{\boldsymbol{\mu}}_{\boldsymbol{w}}^{\text{new}}$ and $\tilde{\boldsymbol{\Sigma}}_{\boldsymbol{w}}^{\text{new}}$
    Compute the segment-wise likelihood $\{p_k^{\text{new}}\}_{k=1:K}$ using Eq.(4.6) and 4.7 under $\tilde{\pi}^{\text{new}}$
    Update θ by taking a gradient step w.r.t. $J(\theta)$ in Eq.(4.9)
  end for
end for

4.5. Experiments

We evaluate the effectiveness of our model through experiments on a variety of simulated robot manipulation tasks. The performance of TCE is compared against well-known deep RL algorithms as well as methods specifically designed for correlated actions and consistent trajectories. The evaluation is designed to answer the following questions:

1. Can our model effectively train the policy across diverse tasks, incorporating various robot types, control paradigms (task and joint space), and reward configurations?
2. Does the incorporation of movement correlations lead to higher task success rates?
3. Are there limitations or trade-offs when adopting our proposed learning strategy?

For the comparative evaluation, we select the following methods: PPO, SAC, TRPL, gSDE, PINK [162] and BBRL. Descriptions, hyper-parameters, and references to the used code bases of these methods can be found in Appendix C.2.1. We use step-based methods (PPO, SAC, TRPL, gSDE, and PINK) to predict the lower-level actions for each task. On the other hand, for episodic methods like BBRL and TCE, we predict position and velocity trajectories and then employ a PD controller to compute the lower-level control commands. Across all experiments, we measure task success in terms of the number of environment interactions required. Each algorithm is evaluated using 20 distinct random seeds. Results are quantified using the Interquartile Mean (IQM) and are accompanied by a 95% stratified bootstrap confidence interval [156].



(a) Success Rate **(b)** Performance Profile

Figure 4.5.: Metaworld Evaluation. (a) Overall Success Rate across all 50 tasks, reported using Interquartile Mean (IQM) [156]. (b) Performance profile, illustrating the fraction of runs that exceed the threshold specified on the x-axis.

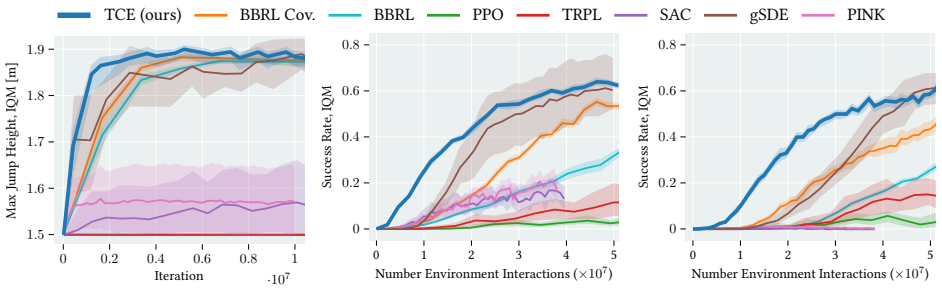
4.5.1. Large-scale Robot Manipulation Benchmark using Metaworld

We begin our evaluation using the Metaworld benchmark [165], a comprehensive testbed that includes 50 manipulation tasks of varying complexity. Control is executed in a 3-DOF task space along with the finger closeness, and a dense reward signal is employed. In contrast to the original evaluation protocol, we introduce a more stringent framework. Specifically, each episode is initialized with a randomly generated context, rather than a fixed one. Additionally, we tighten the success criteria to only consider a task successfully completed if the objective is maintained until the final time step. This adjustment mitigates scenarios where transient successes are followed by erratic agent behavior. While we train separate policies for each task, the hyperparameters remain constant across all 50 tasks. For each method, we report the overall success rate as measured by the IQM across the 50 sub-tasks in Fig. 4.5a. The performance profiles are presented in Fig. 4.5b.

In both metrics, our method significantly outperforms the baselines in achieving task success. BBRL exhibits the second-best performance in terms of overall consistency across tasks but lags in training speed compared to step-based methods. We attribute this difference to the use of per-step dense rewards, which enables faster policy updates in step-based approaches. TCE leverages the advantages of both paradigms, surpassing other algorithms after approximately 10^7 environment interactions. Notably, the off-policy methods SAC and PINK were trained with fewer samples than used for on-policy methods due to their limitations in parallel environment utilization. PINK achieved superior final performance but at the expense of sample efficiency compared to SAC. In Appendices B.1.2 and B.1.3, we provide the results for 50 tasks and a performance profile analysis of TCE.

4.5.2. Joint Space Control with Multi-task Objectives

Next, we investigate the advantages of modeling complete movement correlations and the utility of intermediate feedback for policy optimization. To this end, we enhance the BBRL algorithm by expanding its factorized Gaussian policy to accommodate full covariance (BBRL Cov.), thereby



(a) Hopper Jump, Max Height **(b)** Box Pushing, Dense **(c)** Box Pushing, Sparse

Figure 4.6.: Task Evaluation of (a) Hopper Jump Max Height. (b) Box Pushing success rate in dense reward, and (c) Box Pushing success rate in sparse reward setting.

capturing movement correlations. Both the original and augmented versions of BBRL are included in the subsequent task evaluations. We evaluate the methods on a customized Hopper Jump task, sourced from OpenAI Gym [166]. This 3-DoF task primarily focuses on maximizing jump height while also accounting for precise landing at a designated location. Control is executed in joint space. We report the max jump height as the main metric of success in Fig. 5.4c. Our method exhibits quick learning and excels in maximizing jump height. Both BBRL versions exhibit similar performance, while BBRL Cov. demonstrates marginal improvements over the original. However, they both fall behind TCE in training speed, highlighting the efficiency gains we achieve through intermediate state-based policy updates. Step-based methods like PPO and TRPL tend to converge to sub-optimal policies. The only exception is gSDE. As an augmented step-based method, it enables smoother and more consistent exploration but exhibits significant sensitivity to model initialization (random seeds), evident from the wide confidence intervals.

4.5.3. Contact-rich Manipulation with Dense and Sparse Reward Settings

We further turn to a 7-DoF robot box-pushing task adapted from [10]. The task requires the robot's end-effector, equipped with a rod, to maneuver a box to a specified target position and orientation. The difficulty lies in the need for continuous, correlated movements to both position and orient the box accurately. To amplify the complexity, the initial pose of the box is randomized. We test two reward settings: dense and sparse. The dense setting offers intermediate rewards inversely proportional to the current distance between the box and its target pose, while the sparse setting only allocates rewards at the episode's end based on the final task state. Performance metrics for both settings are shown in Fig. 5.4d and 5.4b. In either case, TCE and gSDE exhibit superior performance but with TCE demonstrating greater consistency across different random seeds. The augmented BBRL version outperforms its original counterpart, emphasizing the need for fully correlated movements in tasks that demand consistent object manipulation. The other step-based methods struggle to learn the task effectively, even when dense rewards are provided. This further

highlights the advantages of modeling the movement trajectory as a unified action, as opposed to a step-by-step approach.

4.5.4. Action Correlations Predicted by Trained Policies

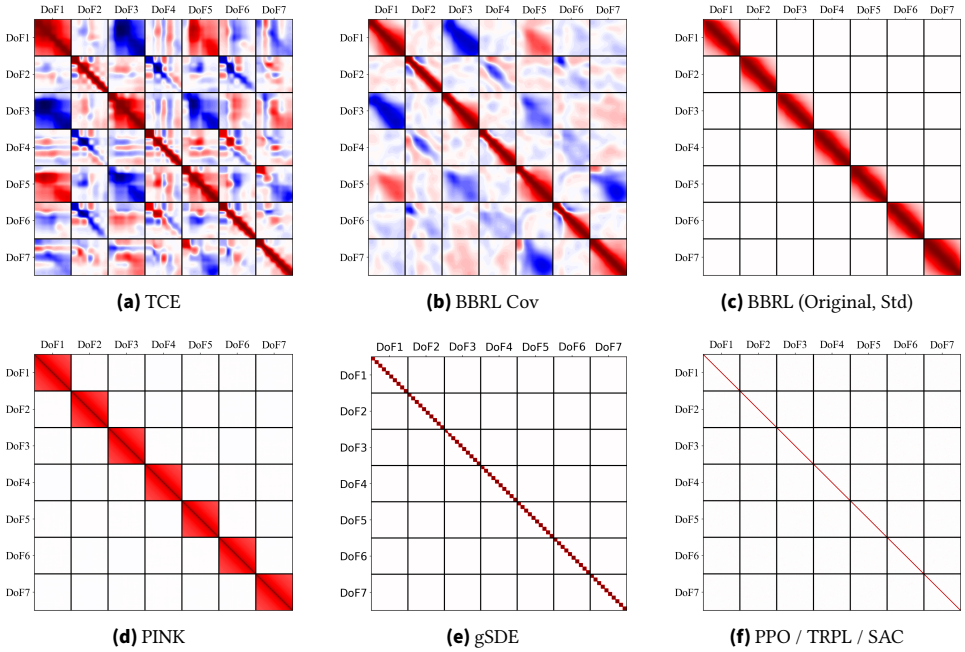
We plot the action correlation coupling DoF and time steps in Fig. 4.7. All policies were trained under the box-pushing task with a dense reward setting. The action outputs for TCE, BBRL, and BBRL Cov are the positions of the robot joints, while step-based methods, such as PPO, predict actions in the torque space. TCE and BBRL Cov demonstrate the ability to predict actions correlated both temporally and across DoF, as indicated by the non-zero off-diagonal elements in their correlation matrices. In contrast, the original BBRL translates a factorized weight distribution into a block-diagonal action correlation matrix, capturing variance within individual DoF but not between them. Similarly, PINK is constrained to modeling intra-DoF correlations, which depend solely on the time difference. This limitation arises from the wide-sense stationarity of the noise, resulting in a constant value along each diagonal. gSDE, however, models temporal correlation but only over a few consecutive time steps, observable along the diagonal elements. Actions predicted by PPO, TRPL, and SAC lack both temporal and DoF correlation, resulting in correlation matrices resembling identity matrices. Interestingly, for methods that only capture intra-DoF correlations, these correlations are uniformly positive. This trend may relate to the control cost in the reward function, promoting consistent movement within each DoF over time. On the other hand, TCE and BBRL Cov are unique in their ability to capture negative correlations, both between and within DoF, enhancing their flexibility in trajectory sampling.

4.5.5. Trajectory Smoothness Evaluation

We compared the trajectory smoothness of all methods in Table 4.1. To ensure a fair comparison, all methods were trained using the fixed start box pushing dense reward setting as originally reported in [10], where each method achieved a minimum 50% success rate. Trajectories for evaluation were generated using the mean prediction of the policy. The smoothness was assessed using three metrics: *maximum jerk*, *mean squared jerk* [167], and *dimensionless jerk* [168]. The first two metrics are standard in robot trajectory generation [169, 170], while the last is proposed as a more equitable measure of smoothness, eliminating the effects of motion magnitude and time scaling. TCE and BBRL Cov outperformed all other methods in smoothness, followed by the original BBRL. This performance disparity likely stems from the original BBRL's inability to model inter-DoF movement correlations. In contrast, all step-based methods exhibited lower smoothness, attributable to their inherent per-step action selection approach.

4.5.6. Hitting Task with High Sparsity Reward Setting

In our last experiment, we assess the limitations of our method using a 7-DoF robot table tennis task, originally from [98]. The robot aims to return a randomly incoming ball to a desired target on



(a) TCE **(b)** BBRL Cov **(c)** BBRL (Original, Std)

(d) PINK **(e)** gSDE **(f)** PPO / TRPL / SAC

Figure 4.7.: This figure presents predicted actions' correlation across 7 DoF and 100 time steps, visualized in a 700x700 correlation matrix. Each 100 $\times$ 100 square tile demonstrates the movement correlation between two DoF during these steps. Correlation values range from -1 (negative correlation, depicted in blue) to 1 (positive correlation, depicted in red), with white areas indicating no correlation. The action outputs for TCE, BBRL, and BBRL Cov are the positions of the robot joints, whereas step-based methods predict actions in the torque space. TCE and BBRL Cov exhibit to a higher capacity of movement correlations. The original BBRL and PINK only model correlations within each DoF. gSDE models correlations over a few consecutive time steps. We show only one representative matrix for PPO, TRPL, and SAC, as their results are visually identical, typically resulting in matrices resembling the identity matrix.

Table 4.1.: Trajectory Smoothness, Mean (Std) of Three Metrics over 400 Trajectories.

Metric	TCE	BBRL Cov	BBRL	PPO	TRPL	gSDE	SAC	PINK
Maximum Jerk, $\times 10^3 \mathrm{rad/s^3}$	**3.4 (1.9)**	3.5 (1.5)	4.3 (1.4)	9.1 (3.3)	12.9 (4.8)	6.9 (2.2)	9.3 (4.2)	6.5 (1.7)
Mean Sq. Jerk, $\times 10^6 \mathrm{rad^2/s^6}$	**0.2 (0.2)**	**0.2 (0.3)**	0.6 (0.6)	1.3 (0.9)	5.5 (8.6)	0.8 (0.6)	3.9 (1.1)	1.7 (0.7)
Dimensionless Jerk, $\times 10^6$	**61 (73)**	64 (56)	128 (49)	141 (67)	555 (472)	122 (83)	506 (262)	311 (127)

the opponent's court. To enhance the task's realism, we randomize the robot's initial state instead of using a fixed starting pose. This task is distinct due to its one-shot nature: the robot has only one chance to hit the ball and loses control over the ball's trajectory thereafter. The need for diverse hitting strategies like forehand and backhand adds complexity and increases the number of samples required for training. Performance metrics are presented in Fig. 4.8. The BBRL Cov. emerges as the leader, achieving a 20% higher success rate than other methods. It is followed by TCE and the

original BBRL, with TCE displaying intermediate learning speeds between the two BBRL versions. Step-based methods, led by TRPL at a mere 15% task success, struggle notably in this setting. We attribute the underperformance of TCE and step-based methods to the task's reward sparsity, which complicates the value function learning of SRL and TCE. Despite these challenges, TCE maintains its edge over other baselines, further attesting to its robustness, even under stringent conditions.

4.6. Conclusion

We introduced TCE that synergizes the exploration advantages of ERL with the sample efficiency of SRL. Empirical evaluation showcases that TCE matches the sample efficiency of SRL and consistently delivers competitive asymptotic performance across various tasks. Furthermore, we demonstrated both the sample efficiency and policy performance of episodic policies can be further improved by incorporating proper correlation modeling. Despite the promise, several opening questions remain for future work. Firstly, TCE yields moderate results for tasks characterized by particularly sparse reward settings, as observed in scenarios like table tennis. Secondly, ERL approaches often need a low-level tracking controller, which might not be feasible for certain task types, such as locomotion. Additionally, the current open-loop control setup lacks the adaptability needed for complex control problems in dynamic environments where immediate feedback and swift adaptation are crucial. These issues will be at the forefront of our future work.

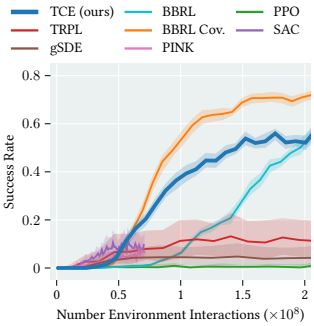


Figure 4.8.: Table Tennis with High reward sparsity.

5. Transformer-based Episodic Reinforcement Learning

Current Episodic RL (ERL) methods are almost exclusively on-policy. They must gather fresh data for every update and cannot reuse past experience, which hurts sample efficiency. Thus **[RQ3]** asks: *How can off-policy techniques be brought into an episodic framework without losing ERL's trajectory-level benefits?*

Contributions. I propose *Transformer-based Off-Policy Episodic RL (TOP-ERL)*, an ERL variant that supports efficient off-policy updates.

1. *Transformer critic.* A causal Transformer estimates state–action values for each trajectory segment (Fig. 2.6). It processes the start state and the sequence of segment actions, capturing long-range temporal dependencies via self-attention.

2. *N-step returns without importance sampling.* The critic enables direct N-step return computation, removing the high variance that importance sampling introduces in standard off-policy methods [45]. This yields stable, low-bias value estimates and speeds up learning.

3. *Empirical gains.* Across diverse manipulation tasks—including sparse-reward and hard-exploration settings—TOP-ERL outperforms both on-policy ERL and leading step-based algorithms. Ablation studies confirm that the Transformer architecture and the N-step return formulation are key to the observed improvements.

These results establish TOP-ERL as a practical route to off-policy sample efficiency while preserving the smooth, temporally coherent behaviours characteristic of episodic control.

The remainder of this chapter has been published as [14]: Ge Li, Dong Tian, Hongyi Zhou, Xinkai Jiang, Rudolf Lioutikov, Gerhard Neumann. "TOP-ERL: Transformer-based Off-Policy Episodic Reinforcement Learning". *In: International Conference on Learning Representation (ICLR),* 2025.

5.1. Introduction

This work proposes a novel off-policy Reinforcement Learning (RL) algorithm that utilizes a transformer architecture for predicting the values for action sequences. These returns are effectively used to update the policy that predicts a smooth trajectory instead of a single action in each decision step. Predicting a whole trajectory of actions is commonly done in episodic RL (ERL) [163] and differs conceptually from conventional step-based RL (SRL) methods like SAC [8] where an action

is sampled in each time step. The action selection concept in ERL is promising as shown in recent works in RL [10, 13]. Similar insights have been made in the field of Imitation Learning, where predicting action sequences instead of single actions has led to great success [171, 172]. Additionally, decision-making in ERL aligns with the human's decision-making strategy, where the human generally does not decide in each single time step but rather performs a whole sequence of actions to complete a task – for instance, swinging an arm to play tennis without overthinking each per-step movement.

Episodic RL is a distinct family of RL that emphasizes the maximization of returns over entire episodes, rather than optimizing the intermediate states during environment interactions [90, 91, 9]. Unlike SRL, ERL shifts the solution search from per-step actions to a parameterized trajectory space, leveraging techniques like Movement Primitives (MPs) [1, 3] for generating action sequences. This approach enables a broader exploration horizon [163], captures temporal and degrees of freedom (DoF) correlations [13], and ensures smooth transitions between re-planning phases [96]. Recent advances have integrated ERL with deep learning architectures, demonstrating significant potential in areas such as versatile skill acquisition [41] and safe robot reinforcement learning [99]. However, despite their advantages, ERL methods often suffer from low update efficiency. Nearly all ERL approaches to date remain constrained to an on-policy training paradigm, limiting their ability to exploit more efficient off-policy update rules, where an action-value function, or *critic*, is explicitly learned to guide policy updates and action selection. The primary challenge is that prominent off-policy methods, such as SAC [8], rely on temporal difference (TD) error [103] to update the critic, which implicitly assumes that actions are selected based on each perceived state, rather than a sequence of actions predicted at the start of the episode, as in ERL approaches. In this work, we address this limitation by predicting the N-step return [45] for a sequence of actions using a Transformer architecture, enabling the learning of sequence values within an off-policy framework.

Transformer in RL. Over the past few years, the Transformer architecture [11] has emerged as one of the most powerful models for sequence data. It has been been integrated into RL across various domains, capitalizing on their strengths in sequence pattern recognition from static datasets and functioning as a memory-based architecture, which aids in task understanding and credit assignment. Applications of Transformers in RL include offline RL [173, 141, 140], offline-to-online fine-tuning [143, 174, 175], handling partially observable states [139, 176, 177], and model-based RL [178]. However, the use of Transformers within a model-free online RL framework, specifically for sequence action prediction and evaluation, remains largely unexplored [179]. This is noteworthy, as similar techniques, such as action chunking [180], have already proven successful in other domains like imitation learning.

In this work, we propose **Transformer-based Off-Policy ERL (TOP-ERL)**, which leverages the Transformer as a critic to predict the value of action sequences. Given a trajectory from ERL, we split it into smaller segments and input them into the Transformer for value prediction. We adapt off-policy update rules for action sequences, using the N-step TD error for critic updates. The policy then selects action sequences based on the preferences of the Transformer critic, similar to SAC. Compared to existing ERL and SRL methods, we show that TOP-ERL improves both policy quality

and sample efficiency, outperforming them in several simulated robot manipulation tasks. **Our contributions** are:

1. A novel off-policy RL method that integrates the Transformer as a critic for action sequences in a model-free, online RL framework.
2. The use of N-step return as the learning objective for the Transformer critic.
3. Comprehensive evaluation on simulated robotic manipulation tasks, demonstrating superior performance against baselines.
4. Analysis of different critic update rules, design choices, and the impact of segment length on model performance.

5.2. Related Work

Episodic RL. The study of ERL approaches dates back to the 1990s. Early approaches employed black-box optimization techniques to update parameters of policies, such as small MLPs [90, 91, 92]. Due to the substantial data requirements of black-box algorithms and the limited computational resources available at the time, these approaches were constrained to low-dimensional tasks like Pendulum and Cart Pole. Subsequent works [93, 94] demonstrated that, given sufficient computational resources, ERL methods can also achieve comparable performance to step-based RL on challenge locomotion tasks, such as Ant and Humanoid, at the cost of more samples for convergence. Another line of research in ERL focuses on more compact policy representations. Peters et al. [9] first proposed using movement primitives (MPs) as parameterized policies for ERL, reducing the search space from the high-dimensional neural network parameter space to the MP weight space, which typically ranges from 20 to 50 dimensions, resulting in less samples required for convergence. Using MPs as policies also provides additional benefits, such as smooth trajectory generation and more consistent exploration [13]. MP-based ERL approaches have demonstrated the ability to master complex manipulation tasks such as robot baseball [9] and juggling [95]. To further improve sample efficiency, [62] introduced a model-based method to enable more sample-efficient black-box searching. However, these methods are limited in handling tasks with contextual variations, e.g., changing goals. To address this limitation, Abdolmaleki et al. [97] and Celik et al. [98] extend MP-based ERL by using linear policies conditioned on context. Otto et al. [10] enhanced contextual MPRL by employing neural network policies and trust-region regularized policy update. Despite these advances, existing ERL methods generally treat the episodic trajectory as a black box. While this approach allows them to handle sparse and even non-Markovian rewards, ignoring the temporal structure within each episode leads to lower sample efficiency compared to step-based methods, especially in settings with dense rewards. To address this issue, a most recently proposed method, *Temporally-Correlated ERL* (TCE) [13] introduced a more efficient update scheme that "opens the black-box" and utilizes sub-segment information for policy update while retaining the benefit of episodic exploration. Although TCE improves the sample efficiency of contextual ERL methods, it still relies on on-policy policy gradient updates, which are considered sample-inefficient. To the

best of our knowledge, TOP-ERL is the first off-policy ERL algorithm capable of handling contextual tasks.

Transformers in model-free RL. Inspired by the success of Transformers in domains requiring sequence reasoning, the study incorporating Transformers in RL to solve tasks that require long-horizon memory emerged. However, using standard Transformers in RL could results in performance comparable to random policy [139]. To address this issue, *Gated Transformer-XL*(GTrXL) [139] augmented Transformer-XL with GRU-style gating layers between multi-head self-attention layers, stabilizing the training of deep Transformer networks (up to 12 layers) with online RL. Another research line focuses on utilizing Transformers to enhance offline RL, where the learning process is based on a fixed dataset collected by arbitrary behavior policies. Decision Transformers [132] were the first to formulate offline RL as a sequence modeling problem. Subsequent works extended this approach by incorporating dynamic history length adjustment [140], Q-learning [141], and replacing the Transformer with a more efficient state-space model [142]. Online Decision Transformers [143] further advanced Decision Transformer by introducing online fine-tuning. In contrast to these studies, which primarily focus on offline RL or fine-tuning pre-trained models, TOP-ERL is designed for online RL and does not rely on offline training. Additionally, TOP-ERL is not designed to solve tasks that require long-horizon memory. Instead, it focuses on using a Transformer-based critic to improve multi-step TD learning within the ERL framework.

5.3. Preliminaries

5.3.1. Off-Policy Reinforcement Learning

Markov decision process (MDP). RL learns policies that maximize cumulative rewards in a given environment, modeled as an MDP. Formally, we consider an MDP defined by a tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where both state $\mathcal{S}$ and action spaces $\mathcal{A}$ are continuous. Here, $P(s'|s,a)$ denotes the state transition probability, $r(s,a)$ is the reward function, and $\gamma \in [0,1]$ is the discount factor. The goal of RL is to find a policy $\pi(a|s)$ that maximizes the expected *return*, which is the sum of discounted future rewards as $G_t(s_t, a_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$.

In **off-policy RL**, the agent learns a policy $\pi(a|s)$ using data generated by a different behavior policy $\pi_b(a|s)$. This enables off-policy methods to reuse past experiences, significantly improving sample efficiency against on-policy methods. A common approach in off-policy RL is to use a *critic*, which estimates the action-value function $Q^\pi(s,a)$ and is updated using a temporal difference (TD) error

$$Q^\pi(s,a) = \mathbb{E}_\pi \left[G_t \mid s_t = s, a_t = a \right], \quad \delta_t = r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t), \tag{5.1}$$

where the TD error δ_t estimates the difference between the current Q-value and the target Q-value. While the above single-step TD error is useful, it can suffer from high bias and slow convergence, especially in environments with delayed rewards. To address this, N-step returns [103] are often used to provide a better balance between bias and variance.

The N-step return extends the single-step TD return by incorporating multiple future time-steps into the target. Unlike bootstrapping after a single time step, the N-step return accumulates rewards over N steps before using the current value estimate for bootstrapping. These estimates are typically less biased than the 1-step return, but also contain more variance. In off-policy settings, the N-step return typically involves importance sampling [45], as the selection of the future action path used to accumulate rewards differs from the current policy $\pi(a|s)$, seen as:

$$G_t^{(N)}(s_t, a_t) = \sum_{n=0}^{N-1} \left(\prod_{j=0}^{n} \rho_{t+j} \right) \gamma^n r_{t+n} + \left(\prod_{j=0}^{N-1} \rho_{t+j} \right) \gamma^N Q^\pi(s_{t+N}, a_{t+N}), \tag{5.2}$$

where $\rho_t = \frac{\pi(a_t|s_t)}{\pi_b(a_t|s_t)}$ is the importance sampling ratio, ensuring that updates remain unbiased even when using trajectories generated by a different policy.

Despite this mathematical correction, applying N-step returns in off-policy learning can face difficulties, particularly for long sequences. The product of importance ratios can become highly volatile, leading to either exploding or vanishing values over extended trajectories, which in turn can cause high variance in the value estimates and destabilize the learning process. In TOP-ERL, however, we employ N-step return for computing the target value of a sequence of actions, i. e. $G_t^{(N)}(s_t, a_t, a_{t+1}, ..., a_{t+N})$, where N-step actions are determined in a sequence read from the replay buffer, rather than sampled from the policy. Therefore, the resulting formulation does not contain the importance weights. We will further discuss the details in Sec. 5.4.3.

5.3.2. Episodic Reinforcement Learning (ERL)

Episodic RL [90, 163] focuses on predicting an entire sequence of actions to complete a task, optimizing the cumulative return without explicitly considering detailed state transitions within the episode. Typically, ERL methods utilize a parameterized trajectory generator, such as motion primitives (MP) [1, 3], which predicts a trajectory parameter vector $\boldsymbol{w}$. This vector is then mapped to a full action trajectory $\boldsymbol{a}(\boldsymbol{w}) = [\boldsymbol{a}_t]_{t=0}$, where T is the trajectory length. Here, $\boldsymbol{a}_t \in \mathbb{R}^D$ denotes the action at time step t, and D represents the dimensionality of the action space, such as the degrees of freedom (DOF) in a robotic system. In this framework, an intelligent agent—such as a robot—executes the predicted action sequence directly as motor commands or follows the trajectory using a tracking controller.

Although ERL predicts an entire action trajectory, it still adheres to the *Markov property*, where the state transition probability depends only on the current state and action [45]. Thus, while the action sequence in ERL spans multiple time steps, the underlying process remains consistent with the MDP formalism. This approach is conceptually related to techniques such as action repeat [157] and temporally correlated action selection [154, 162], which also incorporate temporal dependencies into action selection.

Movement Primitives (MP), as parameterized trajectory generators, play a crucial role in ERL. We briefly highlight key MP methodologies and their mathematical foundations used in this work,

with a more detailed discussion in Chapter 3. Schaal [1] introduced Dynamic Movement Primitives (DMPs), which incorporates a forcing term into a dynamical system to generate smooth trajectories from a given initial condition, such as a robot's position and velocity at a particular time[1].

$$\tau^2 \ddot{y} = \alpha(\beta(g - y) - \tau \dot{y}) + f(x), \quad f(x) = x \frac{\sum \varphi_i(x) w_i}{\sum \varphi_i(x)} = x \boldsymbol{\varphi}_x^\intercal \boldsymbol{w}, \tag{5.3}$$

where $y = y(t)$, $\dot{y} = \mathrm{d}y/\mathrm{d}t$, $\ddot{y} = \mathrm{d}^2 y/\mathrm{d}t^2$ denote the position, velocity, and acceleration of the system at time t, respectively. Constants α and β are spring-damper parameters, with g as the goal attractor and τ as a time constant modulating the speed of trajectory execution. The functions $\varphi_i(x)$ represents the basis functions for the forcing term, and the trajectory's shape is determined by the weight parameters $w_i \in \boldsymbol{w}$, for $i = 1, ..., N$. The trajectory $y_{0:T}$ is typically computed by numerically integrating the dynamical system from the start to the end. Building on the same concepts, Li et al. [12] proposed Probabilistic Dynamic Movement Primitives (ProDMPs), which directly uses the closed-form solution of Eq.(5.3). ProDMP employs a linear basis function representation to directly map a parameter vector $\boldsymbol{w}$ to its corresponding trajectory $y_{0:T}$:

$$y(t) = \boldsymbol{\Phi}(t)^\intercal \boldsymbol{w} + c_1 y_1(t) + c_2 y_2(t). \tag{5.4}$$

Here, the terms $c_1 y_1(t) + c_2 y_2(t)$ ensure precise trajectory initialization, with the constants c_1, c_2 calculated based on the initial condition $y_b, \dot{y}_b$ at time t_b. The term $\boldsymbol{\Phi}(t)$ denotes the integral form of the basis functions $\boldsymbol{\varphi}$ used in the Eq.(5.3). Unlike DMP, ProDMP benefits from the closed-form solution of the dynamic system, enabling faster computation and probabilistic modeling without the burden for numerical integration. This allows for flexible trajectory generation and precise initial condition enforcement. In TOP-ERL, we leverage ProDMP's fast initial condition enforcement to compute accurate target values for the Transformer critic, thereby reducing bias in policy learning.

ERL Learning Objectives. A key distinction between ERL and step-based RL (SRL) lies in the action space. ERL shifts the solution search from the per-step action space $\mathcal{A}$ to a parameterized trajectory space $\mathcal{W}$, predicting the trajectory parameters as $\pi(\boldsymbol{w}|\boldsymbol{s})$. As a result, a trajectory parameterized by $\boldsymbol{w}$ is treated as a single data point in $\mathcal{W}$. This often leads ERL to employ black-box optimization methods for trajectory optimization [93]. The learning objective in ERL is often formulated using an importance sampling ratio, such as in BBRL [10]

$$\text{Update using trajectory parameter:} \quad J = \mathbb{E}_{\pi_{\text{old}}(\boldsymbol{w}|\boldsymbol{s})} \left[\frac{\pi_{\text{new}}(\boldsymbol{w}|\boldsymbol{s})}{\pi_{\text{old}}(\boldsymbol{w}|\boldsymbol{s})} G^{\pi_{\text{old}}}(\boldsymbol{s}, \boldsymbol{w}) \right], \tag{5.5}$$

where π represents the policy parameterized by $\boldsymbol{\theta}$, typically using a neural network. The terms *new* and *old* refer to the current policy being optimized and the policy used for data collection, respectively. The initial state $\boldsymbol{s} \in \mathcal{S}$ defines the starting configuration and objective of the task, serving as input to the policy. The policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{s})$ determines the likelihood of selecting trajectory parameters $\boldsymbol{w}$. The term $G^{\pi_{\text{old}}}(\boldsymbol{s}, \boldsymbol{w}) = \sum_{t=0}^{T} \gamma^t r_t$ represents the return accumulated by executing

[1] An initial condition in mathematics refers to the value of a function or its derivatives at a starting point, which can be specified at any time and is not necessarily at $t = 0$.

the trajectory under an old policy, where γ is the discount factor and r_t is the reward at time step t. By leveraging parameterized trajectory generators like MPs, ERL benefits from consistent exploration, smooth action trajectories, and improved robustness against local optima, as highlighted by Otto et al. [10]. To further enhance learning efficiency, recent work TCE [13] proposes a hybrid update strategy that decomposes the trajectory parameter-wise update into the segment-wise updates, incorporating per-step information into ERL's learning objective. This approach divides the longer action trajectory into smaller segments, calculating the return of each segment. The new learning objective adapts Eq.(5.5), with the maximization of segment-wise returns as

$$\text{Update using segments:} \quad J = \mathbb{E}_{\pi_{\text{old}}(\boldsymbol{w}|\boldsymbol{s})}\left[\frac{1}{K}\sum_{k=1}^{K}\frac{p^{\pi_{\text{new}}}([\boldsymbol{a}_t^k]_{t=0:L}|\boldsymbol{s})}{p^{\pi_{\text{old}}}([\boldsymbol{a}_t^k]_{t=0:L}|\boldsymbol{s})}\, G^{\pi_{\text{old}}}(\boldsymbol{s}_0^k, [\boldsymbol{a}_t^k]_{t=0:L})\right], \tag{5.6}$$

where K and L represent the number and length of the trajectory segments, respectively, with $K = 25$ in the original paper and $k = 1, ..., K$ denotes the segment index. In this expression, p^{π} denotes the likelihood of reproducing the segment, calculated using the parameterized policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{s})$, and $G(\boldsymbol{s}_0^k, [\boldsymbol{a}_t^k]_{t=0:L})$ represents the return of executing the k-th action sequence segment $[\boldsymbol{a}_t^k]_{t=0:L}$ from the segment's starting state $\boldsymbol{s}_0^k$. It is worth noting, despite the usage of importance sampling, both Eq. (5.5) and Eq. (5.6) still remain within the on-policy RL framework. In TOP-ERL, we employ a similar strategy in splitting a long action trajectory into smaller segments, and use these segments for efficient critic and policy updates, under an off-policy framework.

5.4. Transformer-based Off-Policy ERL

In this section, we present TOP-ERL, an innovative off-policy solution for ERL that leverages a Transformer for action sequence evaluation. The section is structured as follows: Section 5.4.1 introduces the Gaussian policy modeling and action trajectory generation, followed by the design of the transformer critic in Section 5.4.2. The learning objectives for the critic and policy are detailed in Section 5.4.3 and Section 5.4.4, respectively, with additional technical details. Lastly, we summarize other design choices in Section 5.4.5. The main contributions of our model are described from Section 5.4.2 to Section 5.4.4, while the remaining sections cover techniques adopted from the literature.

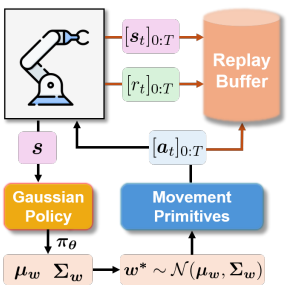


Figure 5.1.: Trajectory generation and environment rollout.

5.4.1. Trajectory Generation: Techniques Adopted from ERL Literature

TOP-ERL adopts a policy structure similar to previous ERL approaches, such as BBRL [10]. As shown in Fig. 5.1, our policy is modeled as a Gaussian distribution, $\pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{s}) = \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}})$, where $\boldsymbol{s}$ defines the initial observation and the task objective, and $\boldsymbol{w}$ represents the parameters of

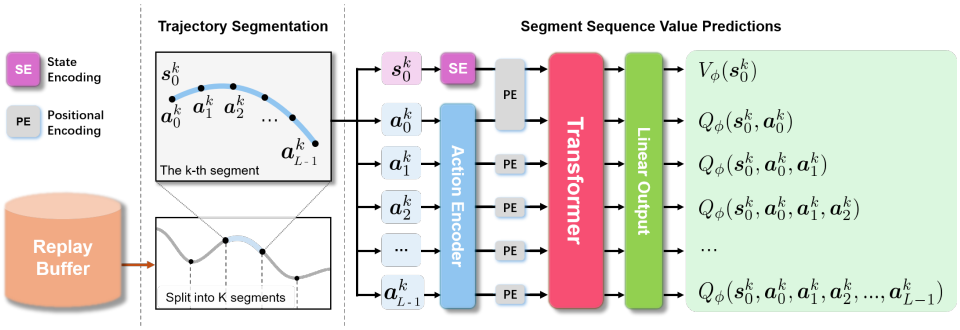


Figure 5.2.: Architecture overview of the Transformer critic, as described in Sec. 5.4.2.

the movement primitive (MPs). In TOP-ERL, we employ ProDMPs [12] to help correct the target computation via enforcing the initial condition of the MP, as discussed later in Section 5.4.3.1. Given an initial task state $\boldsymbol{s}$, the policy predicts the Gaussian parameters and samples a parameter vector $\boldsymbol{w}^*$. This vector is then passed into the movement primitive to generate the action trajectory $[\boldsymbol{a}_t]_{t=0:T}$. The agent then executes the action trajectory in the environment until the end of the episode. During the rollout, both the state trajectory and the reward trajectory are recorded. These, along with the action trajectory, are subsequently stored in the replay buffer $\mathcal{B}$ for later use.

5.4.2. Transformers as value predictor for action sequences

An architectural overview of our Transformer critic is depicted in Fig. 5.2. At each iteration, we sample a batch B of trajectories from the replay buffer and split each trajectory into K segments, where each segment is L time steps long. An ablation on how to select the segment length L can be found in Sec. 5.5.3. The transformer-based critic has $L + 1$ input tokens that are given by each action in the segment $[\boldsymbol{a}_t^k]_{t=0:L-1}$ and the starting state $\boldsymbol{s}_0^k$ of the corresponding segment. These tokens are first processed by corresponding state and action encoders, each modeled by a single linear layer. Positional information is added to the processed tokens through a trainable positional encoding, with $\boldsymbol{s}_0^k$ and the first action token $\boldsymbol{a}_0^k$ sharing the same positional encoding (both at $t = 0$). The tokens are subsequently fed into a decoder-only Transformer, followed by a linear output layer, producing $L + 1$ output tokens. The first output represents the state value $V(\boldsymbol{s}_0^k)$ for the starting state, while the remaining outputs correspond to the state-action values for the subsequent action sequence. For example, $Q(\boldsymbol{s}_0^k, \boldsymbol{a}_0^k, \boldsymbol{a}_1^k, \boldsymbol{a}_2^k)$ represents the value of executing the actions $\boldsymbol{a}_0^k, \boldsymbol{a}_1^k, \boldsymbol{a}_2^k$ sequentially from the starting state $\boldsymbol{s}_0^k$ and subsequently following policy π. A causal mask is applied in the Transformer to ensure that actions do not attend to future steps.

5.4.3. N-step Returns as the target for Transformer Critic

For each predicted state-action value $Q(\boldsymbol{s}_0, \boldsymbol{a}_0^k, ..., \boldsymbol{a}_{N-1}^k)$ we utilize the N-step return as its target. The objective to update the parameters ϕ of the critic is the N-step squared TD error[2]

$$\textbf{Critic loss:} \quad \mathcal{L}(\phi) = \frac{1}{L}\sum_{N=1}^{L-1}\Bigg[\underbrace{Q_\phi(\boldsymbol{s}_0^k, \boldsymbol{a}_0^k, ..., \boldsymbol{a}_{N-1}^k)}_{\text{Predicted value of N actions}} - \underbrace{G^{(N)}(\boldsymbol{s}_0^k, \boldsymbol{a}_0^k, ..., \boldsymbol{a}_{N-1}^k)}_{\text{Target using \textbf{N-step return}}}\Bigg]^2$$
$$+\Bigg[\underbrace{V_\phi(\boldsymbol{s}_0^k)}_{\text{Predicted state value}} - \mathbb{E}_{\tilde{\boldsymbol{w}}\sim\pi_\theta(\cdot|s)}\underbrace{[Q_{\phi_{\text{tar}}}(\boldsymbol{s}_0^k, \tilde{\boldsymbol{a}}_0^k, ..., \tilde{\boldsymbol{a}}_{L-1}^k)]}_{\text{Target of new actions using } \tilde{\boldsymbol{w}}}\Bigg]^2, \tag{5.7}$$

$$\textbf{N-step return:} \quad G^{(N)}(\boldsymbol{s}_0^k, \boldsymbol{a}_0^k, ..., \boldsymbol{a}_{N-1}^k) = \underbrace{\sum_{n=0}^{N-1}\gamma^n r_n}_{\text{N-step rewards}} + \underbrace{\gamma^N V_{\phi_{\text{tar}}}(\boldsymbol{s}_N)}_{\text{Future return after N-step}}. \tag{5.8}$$

Here, $N \in [1, L-1]$ represents the number of actions in a sub-sequence starting from $\boldsymbol{s}_0^k$. The term $Q_{\phi_{\text{tar}}}(\boldsymbol{s}_0^k, \tilde{\boldsymbol{a}}_0^k, ..., \tilde{\boldsymbol{a}}_{L-1}^k)$ in Eq.(5.7) denotes the target value of $V_\phi(\boldsymbol{s}_0^k)$ with actions $\tilde{\boldsymbol{a}}_0^k, ..., \tilde{\boldsymbol{a}}_{L-1}^k$ generated by new MP parameters $\tilde{\boldsymbol{w}}$ sampled from the current policy, $\tilde{\boldsymbol{w}} \sim \pi_\theta(\cdot\,|\boldsymbol{s})$. The term $V_{\phi_{\text{tar}}}(\boldsymbol{s}_N)$ in Eq.(5.8) represents the future return after N steps. Both $Q_{\phi_{\text{tar}}}$ and $V_{\phi_{\text{tar}}}$ are predicted by a target critic [68], with a delayed update rate $\rho = 0.005$. Please note that $Q_{\phi_{\text{tar}}}$ and $V_{\phi_{\text{tar}}}$ are the same transformer network, with and without action tokens.

In off-policy RL literature, there are several alternatives to replace $V_{\phi_{\text{tar}}}(\boldsymbol{s}_N)$ in Eq.(5.8). However, we find that this choice alone performs well in our experiments. In other words, TOP-ERL does not necessarily rely on some common off-policy techniques, such as the clipped double-Q [89], to be stable and effective. We attribute this to the usage of the N-step returns, which help reduce value estimation bias. In Sec. 5.5.3, we show that our model can be further improved using these augmentations, though at a cost of additional computation.

Unlike Eq.(5.2), our N-step return targets $G^{(N)}(\boldsymbol{s}_0^k, \boldsymbol{a}_0^k, \ldots, \boldsymbol{a}_{N-1}^k)$ in Eq.(5.8) do not include importance sampling as the the action sequence $\boldsymbol{a}_0^k, \ldots, \boldsymbol{a}_{N-1}^k$ is directly used as input tokens for the Q-function. Hence, the actions are fixed and we do not require to compute any expectations over the current policy's action selection. Hence, using the fixed action sequence in Eq.(5.8) as input to the Q-Function eliminates the need for importance sampling, thus avoiding the high variance typically introduced by it in off-policy methods, as discussed in Sec. 5.3.1.

5.4.3.1. Enforce initial condition for newly predicted action sequence

When calculating the target value $Q_{\phi_{\text{tar}}}(\boldsymbol{s}_0^k, \tilde{\boldsymbol{a}}_0^k, ..., \tilde{\boldsymbol{a}}_{L-1}^k)$ in Eq.(5.7), a new parameter vector is sampled from the current policy $\tilde{\boldsymbol{w}} \sim \pi_\theta(\,\cdot\,|\boldsymbol{s})$, generating a new action trajectory $[\tilde{\boldsymbol{a}}_t]_{t=0:T}$,

[2] For simplicity, we omit the expectation over buffer $\mathcal{B}$ and average over segment number K in Eq.(5.7).

with $[\tilde{\boldsymbol{a}}_t^k(\tilde{\boldsymbol{w}})]_{t=0:L-1}$ as a sub-sequence. However, this sequence is not necessarily guaranteed to pass through the segment's starting state $\boldsymbol{s}_0^k$, which creates a mismatch between the state and corresponding action sequence when querying the target in Eq.(5.7).

To address this issue, we append the old reference position to $\boldsymbol{s}_0^k$, and then leverage the dynamic system formulation inherent in ProDMPs by setting the initial condition of the new action sequence to match the old reference at $\boldsymbol{s}_0^k$, as illustrated in Fig. 5.3. The resulting action sequence $[\tilde{\boldsymbol{a}}_t^k(\tilde{\boldsymbol{w}}, \boldsymbol{s}_0^k)]_{t=0:L-1}$ is therefore depending on both the MP parameters $\tilde{\boldsymbol{w}}_\theta(\boldsymbol{s})$ and the initial condition $\boldsymbol{s}_0^k$. This approach is mathematically equivalent to resetting the initial conditions of an ordinary differential equation (ODE), ensuring consistency between the state and action sequences. Further mathematical details and illustration are provided in Appendix C.1.2.

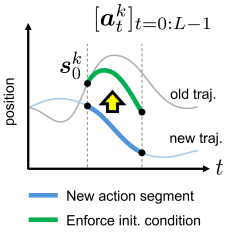


Figure 5.3.: Enforce action initial condition

5.4.4. Policy updates using the Transformer critic

We utilize the transformer critic to guide the training of our policy, using the reparameterization trick similar to that introduced by SAC [8]. The learning objective is to maximize the expected value of the averaged action sequence over varying lengths, defined as:

$$\textbf{Policy Objective:} \quad J(\theta) = \mathbb{E}_{\boldsymbol{s}\sim B}\mathbb{E}_{\tilde{\boldsymbol{w}}\sim\pi_\theta(\cdot|\boldsymbol{s})}\left[\frac{1}{KL}\sum_{k=1}^{K}\sum_{N=0}^{L-1} Q_\phi(\boldsymbol{s}_0^k, \left[\tilde{\boldsymbol{a}}_t^k\right]_{t=0:N})\right], \tag{5.9}$$

where $[\tilde{\boldsymbol{a}}_t^k]_{t=0:N}$ denotes the new action sequence generated by the new MP parameters $\tilde{\boldsymbol{w}}_\theta \sim \pi_\theta(\cdot|\boldsymbol{s})$. This learning objective allows the policy $\pi_\theta(\boldsymbol{w}|\boldsymbol{s})$ to be trained based on the *value preferences* provided by the Transformer critic. We refer Appendix C.1.1 for more detailed discussion.

5.4.5. Additional Design Choices from the Literature for Stable Learning

We summarize the key learning steps in Algorithm4. To effectively capture a broader range of correlations in both temporal and DoF movements, we utilize a full covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{w}}$ in the Gaussian policy [13]. Since the Gaussian policy over MP parameters is typically high-dimensional, we employ the Trust Region Projection Layer (TRPL) [59] for stable policy updates, following the design of previous ERL methods [10, 13]. Detailed discussions are provided in Chapter 2.2.4 and 4.3.4. For the Transformer critic, we apply Layer Normalization [123] as the sole data normalization technique, while disabling dropout, as we found it detrimental to performance. In our experiments, we identified the segment length L as a key hyperparameter. The best results were achieved by randomly sampling L at each update iteration, which we attribute to the Transformer critic's ability to attend to different time horizons, resulting in more robust outcomes.

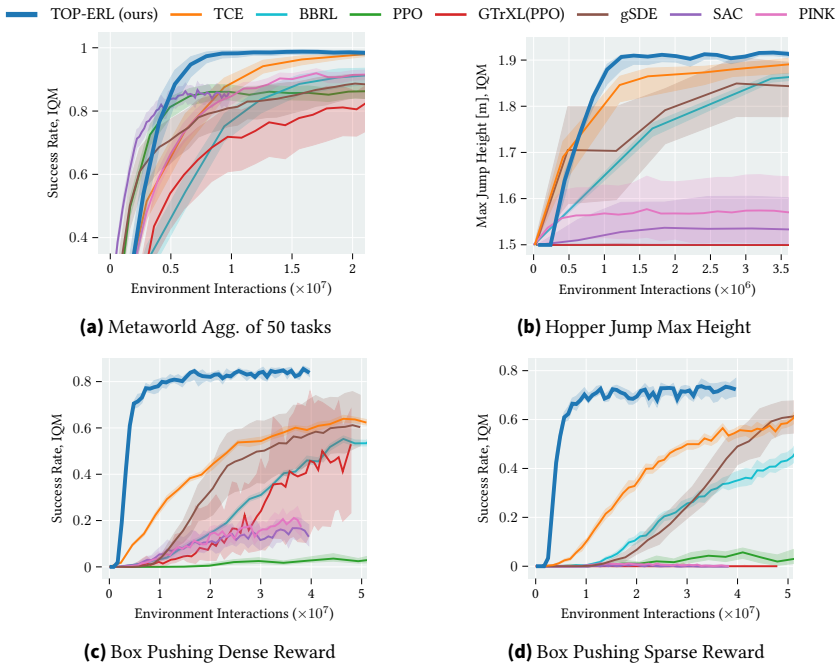
Algorithm 4 Transformer-based Off-Policy Reinforcement Learning (TOP-ERL)

Initialize critic ϕ; target critic $\phi_{\text{tar}} \leftarrow \phi$
Initialize policy θ and replay buffer $\mathcal{B}$
repeat
    Reset environment and get initial task state $\boldsymbol{s}$
    Predict the policy mean $\boldsymbol{\mu_w}$ and covariance $\boldsymbol{\Sigma_w}$
    Sample $\boldsymbol{w}*$ and generate action trajectory $[\boldsymbol{a}]_{0:T}$
    Execute the action trajectory till task ends.
    Store the visited states $[\boldsymbol{s}]_{0:T}$, rewards $[r]_{0:T}$, and the action $[\boldsymbol{a}]_{0:T}$ trajectories in replay buffer $\mathcal{B}$
    for each update step **do**
        From $\mathcal{B}$, sample a batch of $\boldsymbol{s}$, $\boldsymbol{a}$, r trajectories.
        Split them into K segments, each L time steps.
        Compute N-step return targets as in Eq.(5.8)
        Update transformer critic, using Eq.(5.7)
        Update policy, using Eq.(5.9)
    end for
    Update target critic $\phi_{\text{tar}} \leftarrow (1-\rho)\ \phi_{\text{tar}} + \rho\ \phi$
until converged

5.5. Experiments

Our experiments focus on the following questions: I) Can TOP-ERL improve sample efficiency in classical ERL tasks featured by challenging exploration problems? II) How does TOP-ERL perform in large-scale, general manipulation benchmarks? III) How do key design choices affect the performance of TOP-ERL? We compare TOP-ERL against a set of strong baselines. For the ERL comparisons, we select **BBRL** and **TCE** as SoTA ERL methods. For step-based RL, we use **PPO** [7] (on-policy) and **SAC** (off-policy) as established baselines. Additionally, we employed **gSDE** and **PINK**, two step-based RL methods that augment with consistent exploration techniques, to test the impact of exploration strategies. To assess the impact of using Transformer-based architectures in RL, we include **GTrXL** as baseline for online RL with Transformers architecture. It is worth noting that in the original work, GTrXL was trained using VMPO. However, since the original code was not open-sourced, we used the implementation from [181], where GTrXL is trained with PPO instead. For all ERLs, the trajectories are generated using ProDMPs with the same hyperparameters and tracked with PD-controllers (or P-controller for MetaWorlds); for all the SRLs, the action outputs are torque (or delta position for MetaWorlds). An overview of the baselines can be find in Table C.1, and details regarding the implementation and hyperparameters are provided in the Appendix C.4.

The evaluation of TOP-ERL are structured in three phases. First, we demonstrated that TOP-ERL significantly improve the sample efficiency over state-of-the-art ERL methods, showcasing its ability to better handle the challenges of sparse rewards and difficult exploration scenarios [13]. Next, we evaluate TOP-ERL on the Meta-World MT50 [165] benchmark, a large-scale suite of general manipulation tasks. In this setting, TOP-ERL consistently outperform all baselines, demonstrated TOP-ERL's ability to generalize across a wide range of manipulation tasks. Finally, we conduct a



(a) Metaworld Agg. of 50 tasks

(b) Hopper Jump Max Height

(c) Box Pushing Dense Reward

(d) Box Pushing Sparse Reward

Figure 5.4.: Task Evaluation of (a) Metaworld success rate of 50 tasks aggregation. (b) Hopper Jump Max Height. (c) Box Pushing success rate in dense reward, and (d) sparse reward settings.

comprehensive ablation study to analysis which ingredient accounts for the strong performance of TOP-ERL. The results confirm that theses components are essential to achieving the strong performance observed with TOP-ERL. To ensure a robust evaluation, all empirical results are reported using Interquartile Mean (IQM), accompanied by a 95% stratified bootstrap confidence interval [156] across 8 random seeds.

5.5.1. Improving Sample Efficiency in Tasks with Challenging Exploration

ERL methods are renowned for their superior exploration abilities, which often give them an advantage over step-based methods in environments with exploration challenges. However, ERL algorithms are also notoriously sample inefficient, limiting their applicability in scenarios where obtaining samples is expensive. In this evaluation, we investigate whether TOP-ERL can address this limitation by comparing it with baselines on three challenging tasks from Li et al. [13]: HopperJump, a sparse-reward environment where the objective is to maximize the jump height within an episode, and two variants of a contact-rich Box Pushing task. We evaluate the Box Pushing task under both dense and sparse reward settings. Further details about the environments and rewards can be found in Appendix B.1 and C.2. The results of these experiments, shown in Fig. 5.4, demonstrate that TOP-ERL achieved the highest final performance across all three tasks. Notably, in the dense-reward

Box Pushing task, TOP-ERL reached an 80% success rate after just 10 million samples, while the second-best method, TCE, only reaches 60% success after 50 million samples. Similar results is observed in the sparse-reward Box Pushing task, where TOP-ERL reaches 70% success rate with 14 million environment interactions, while TCE and gSDE require 50 million samples to reach 60% success. GTrXL performs moderately in the dense-reward setting, achieving a 50% success rate, but fails completely in the sparse-reward environment. Step-based methods like SAC, PINK and PPO failed in both cases, underscoring the difficulty of these tasks. Among the step-based algorithms, only gSDE achieved comparable performance in compare with ERL methods in these three environments, which we attribute to its state-dependent exploration strategy.

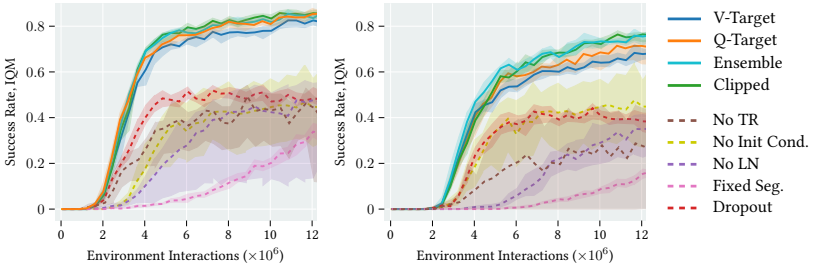
5.5.2. Consistent Performance in large-scale Manipulation Benchmarks

In the previous evaluation, we demonstrated that TOP-ERL significantly improves sample efficiency compared to state-of-the-art ERL baselines, while maintaining strong performance in tasks with challenge exploration. In this evaluation, we focus on answering the second question: How does TOP-ERL perform on standard manipulation benchmarks with dense rewards? We conducted experiments on the Meta-World benchmark[165], reporting the aggregated success rate **across 50 tasks** in the MT50 task set. To ensure a fair comparison, we followed the same evaluation protocol described in [10] and [13], where an episode is only considered successful if the success criterion is met at the end of the episode, a more rigours measure than the original setting where success at any time step counts. The results in Fig. 5.4a show that TOP-ERL achieved highest asymptotic success rate (98%) after 10 million samples. TCE was able to achieve the same success rate but required 20 million interactions. SAC also converged after 10 million samples but with a significantly lower success rate of 85%. BBRL and other step-based methods achieved moderate success rate but required significantly more samples.

5.5.3. Ablation Study and Discussion

Single Q-Network leads to stable and efficient training. We compare four common design choices for targets calculation in Q-function update in Eq.(5.8): 1) V-Target which uses a single V target network, 2) Q-Target, which employs single Q target network, 3)V-Ensemble, which consists of an ensemble of predictions from two V target networks, 4) V-Clip, which takes the minimum of two V target networks. Detailed description of these design choices can be found in Table 5.1. Fig. 5.5 presents the learning curves for TOP-ERL in dense-reward (5.5a) and sparse-reward (5.5b) Box Pushing, while Table 5.1 presents the numerical success rate and computation times per update. The results demonstrate that using a single V target network yields performance comparable to approaches that rely on two target networks, with additionally benefit of significantly reduced computation time (approximately 50% faster). We attribute the stable performance with single target network to the use of N-step Bellman equation in target calculation, as discussed in Sec. 5.4.3.

Key Components Ablation. We evaluate the impact of five key components on the performance of TOP-ERL: trust region constraints in policy updates, enforcing the initial condition at each segment,



(a) Box Pushing Dense Reward **(b)** Box Pushing Sparse Reward

Figure 5.5.: Performance of different critic update strategies (solid lines) and model ablations (dashed lines), using Box pushing dense and sparse reward settings respectively.

Table 5.1.: Quantitative performance and update time of different critic update strategies. With additional computational cost, TOP-ERL can be further enhanced.

Variant	Math	Description	# critic	Time s / iter ↓	Dense Success, % ↑	Sparse Success, % ↑
V-Target (default)	$V_\phi^{\text{tar}}(\boldsymbol{s}_N)$	State value after N steps	1	**1.55**	82.0±2.6	65.7±4.0
Q-Target	$Q_\phi^{\text{tar}}(\boldsymbol{s}_N, \boldsymbol{a}_N, ...)$	Action value after N steps	1	2.44	**86.1 ± 2.7**	69.1 ± 7.5
V-Ensem.	Avg.$(\cdot,\cdot)$	Mean of ≥ 2 target critics	2	2.49	83.8 ± 3.1	**75.7 ± 4.4**
V-Clip	Min$(\cdot,\cdot)$	Minimum of 2 target critics	2	2.49	86.0 ± 3.2	75.5 ± 3.7

the presence of layer normalization, fixed vs. random segment lengths, and the inclusion of dropout in Transformer layers. These evaluations were conducted in both dense-reward and sparse-reward Box Pushing environments using 8 random seeds. The results, presented in Fig. 5.5 as dashed lines, show performance for TOP-ERL with corresponding component been added or removed. The results indicate that the random segment length has the most significant effect on TOP-ERL's performance. When using fixed 25 segments the success rate dropped from 80% to 35% in the dense-reward setting, and from 70% to 20% in the sparse-reward setting. Layer normalization, trust region constraints, and enforcing initial conditions also contributed positively to the performance. Interestingly, adding even a small dropout rate (0.05 in the ablation) had negative impacts on the performance in both tasks. We hypothesize that this effect may be attributed to the use of a relatively small replay buffer combined with a higher buffer update ratio (0.1% in our setting), which likely mitigates the risk of overfitting in Q-function learning, thereby diminishing the benefit of dropout.

Impact of Random Segment Lengths. As shown in the previous ablation study, random segment length played a crucial role in the strong performance of TOP-ERL. To further examine whether this conclusion holds for different segment lengths, we evaluated the dense-reward Box Pushing task with segment lengths ranging from 5% of the episode length to 100% (i.e., no segmentation). The results in Fig. 5.6 indicate that fixed-length segmentation leads to significant performance variation depending on the segment length. In contrast, random segment lengths consistently achieve faster

convergence and higher asymptotic performance. We infer that using a variety of action sequence lengths regularizes the training of the critic network. For example, in Eq. (5.7), the expectation of the Q-value over L actions is used as the target for the V-function's prediction. When L varies across update iterations, the V-function is trained on Q-values derived from different amounts of actions. Additionally, random segmentation simplifies hyperparameter tuning, making it a practical choice. Therefore, we adopt random segmentation length as the default setting for TOP-ERL.

5.6. Conclusion

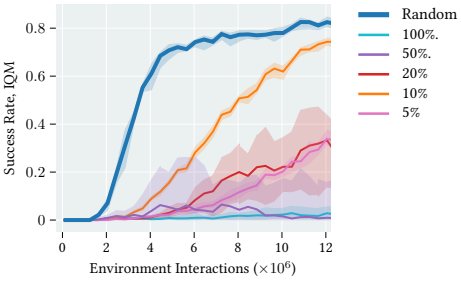


Figure 5.6.: Random or Fixed segment length

This work introduced Transformer-based Off-Policy Episodic RL (TOP-ERL), a novel off-policy ERL method that leverages Transformers for N-steps return learning. By integrating ERL with an off-policy update scheme, TOP-ERL significantly improves the sample efficiency of ERL methods while retaining their advantages in exploration. The use of a Transformer-based critic architecture allows TOP-ERL to bypass the need for importance sampling in N-steps target calculation, stabilizing training while enjoying the benefit of low-bias value estimation provided by N-steps return. TOP-ERL has demonstrated superior performance compared to state-of-the-art ERL approaches and step-based RL methods augmented with exploration mechanism across 53 challenging tasks, providing strong evidence for its broader applicability to wide range of problems. The ablation studies reveal the reasons behind design choices and components, providing insights into the factors contributing to the strong performance of TOP-ERL.

Limitations and Future Works. Despite all the advantages, TOP-ERL shares a limitation common to ERL methods: it generates trajectories only at the start of each episode, making it incapable of handling tasks involving dynamic or target changes within an episode. A promising future research direction would be to incorporate replanning capabilities into TOP-ERL. Additionally, although TOP-ERL uses Transformers as critic, it is not designed to address POMDPs, as the Transformer is used for action-to-go processing in Q-function learning, rather than incorporating state sequences as input. Merging these two paradigms and enhancing TOP-ERL with the ability to handle POMDPs presents another avenue for future investigation.

6. Conclusion

In this dissertation, we proposed methods for modeling trajectories using Movement Primitives (MPs) and Episodic Reinforcement Learning (ERL). We conclude by summarizing the main contributions and discussing their relation to the research questions posed at the beginning of this dissertation, followed by an outlook for future directions.

6.1. Summary

We began by identifying three key challenges in modeling and learning MP trajectories, formulated as the following research questions:

[RQ1] How can dynamical smoothness and probabilistic trajectory modeling be effectively unified within a single MP framework?

[RQ2] How can fine-grained step-based information be integrated into episodic reinforcement-learning frameworks to improve learning efficiency without sacrificing trajectory coherence?

[RQ3] How can modern sequence architectures—particularly Transformers—facilitate data-efficient, off-policy updates in episodic RL when tackling complex robotic tasks?

In Chapter 3, we presented ProDMPs, a unified framework that fuses dynamical and probabilistic MPs. By analytically solving the ODE of the dynamical system, ProDMPs recover a linear basis-function representation for trajectories. This enables explicit handling of initial conditions defined by the robot's current position and velocity, as well as smooth trajectory generation when replanning. We further developed a neural aggregation model for non-linear iterative conditioning, enabling learning of full trajectory covariances with fewer computational resources. We thereby address RQ1. The resulting deep-embedded ProDMPs combine smoothness, goal convergence, trajectory correlation modeling, non-linear conditioning, and online replanning in a single unified model.

In Chapter 4, we introduced TCE, which combines the exploration benefits of ERL with the sample efficiency of SRL. The policy explores in the MP parameter space, predicting the parameters of a Gaussian policy, which are then translated into trajectory distributions. To improve policy-update efficiency, we replace trajectory-level likelihoods with segment-wise likelihoods, approximated using only the two boundary data points for each segment. These segment-wise likelihoods are weighted by corresponding segment-wise advantages, akin to the policy-gradient formulation. This decomposition breaks the "one trajectory = one data point" limitation of previous ERL methods, yielding much finer update granularity and avoiding the inefficiency of black-box optimization.

The hybrid exploration–update strategy of TCE addresses RQ2 by maintaining episodic exploration while greatly enhancing learning efficiency.

Finally, Chapter 5 presented TOP-ERL, a novel off-policy ERL algorithm that leverages a Transformer-based critic for N-step action-value estimation. By combining ERL with an off-policy update scheme, TOP-ERL improves sample efficiency without losing the exploration advantages of ERL. The Transformer critic models long-range temporal dependencies within segmented trajectories and enables N-step target computation without importance sampling. This eliminates the high variance typically associated with off-policy N-step methods, stabilizing training while retaining the low-bias benefits of multi-step returns. Thanks to its efficiency and stability, TOP-ERL successfully addresses RQ3.

6.2. Limitations and Outlook

Movement Primitives Movement Primitives (MPs) have been a fundamental tool for trajectory representation and robot control for more than two decades. With the emergence of deep neural networks, their role has shifted from being the primary reasoning and control mechanism to serving as a structure-imposing output layer that ensures smoothness in trajectories generated by large models. This dissertation demonstrated their strengths in three main aspects: the ability to generate smooth, temporally coherent trajectories with low jerk; the capability to replan gracefully, ensuring continuous transitions between two re-planned motions; and the facilitation of efficient exploration in reinforcement learning by operating in a compact, low-dimensional weight space rather than a high-dimensional action space.

Despite these advantages, we also identified several limitations. Policy updates and sample efficiency remain restricted because MPs predict entire sequences instead of individual step-level actions, limiting fine-grained credit assignment. Furthermore, the number of hyper-parameters—such as basis-function width, number, and overlap—can be large and requires careful tuning, especially for complex tasks. These challenges become more pronounced when using our Probabilistic Dynamic Movement Primitives (ProDMPs), which, while unifying dynamical and probabilistic modelling, still present structural issues.

One notable limitation is the asymmetry of the position and velocity basis functions in ProDMPs, as shown in Fig. 3.1. These functions tend to cluster near the start of the canonical system, leading to unbalanced representational capacity across different phases of a trajectory. As a consequence, ProDMPs often require more basis functions—and thus more hyper-parameters—than simpler models such as ProMPs to achieve comparable accuracy. Another limitation stems from their formulation as a second-order ODE system, which allows explicit specification of only the initial position and velocity. Higher-order via-point constraints are not supported, and convergence to the goal is asymptotic, meaning that exact target attainment within a finite horizon is not guaranteed. Recent work has proposed replacing the DMP canonical system with B-spline bases [182, 99, 183, 184], enabling symmetric basis shapes, richer via-point handling, and stronger terminal guarantees. An illustration of B-spline bases function, originally reported in [184], can be found in Figure 6.1

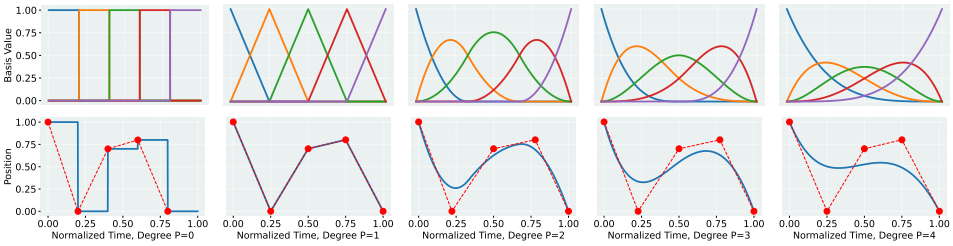


Figure 6.1.: From left to right: Clamped B-Spline basis functions[185] with $P = 0, 1, 2, 3, 4$ (top) and their corresponding generated trajectories (bottom). All trajectories start exactly at the first control point and end at the last control point. A higher B-Spline degree results in smoother trajectories and imposes more initial conditions (position, velocity, acceleration and jerk etc.) at both the start and end. Compared to DMPs and ProDMPs, this representation offers greater via-point flexibility, and the basis functions are symmetric with consistent representational capacity. This figure was originally reported in [184].

Building on such ideas could offer promising directions for overcoming the current representational and control limitations of ProDMPs.

Episodic Reinforcement Learning The two episodic RL algorithms proposed in this dissertation—TCE and TOP-ERL—significantly improved sample efficiency and policy quality compared to existing ERL methods. However, all evaluations were conducted on episodic robot-manipulation tasks with finite time horizons. Many important robotics domains, such as navigation, legged locomotion, and aerial robotics, involve *infinite-horizon* settings where tasks may span minutes, hours, or even days and cannot be solved by a single pre-planned decision. Although MP-based re-planning approaches, such as [96], partially address this limitation by updating policies online, they remain limited in their ability to handle diverse subtasks and to concatenate multiple MP-generated trajectories in a coherent and adaptive manner. In imitation learning, recent work such as [184] has replaced standard action chunking with MP sub-trajectories, offering a promising direction that could be adapted for RL to improve task composition and skill sequencing.

A natural next step is to extend ERL to infinite-horizon tasks by designing policies capable of recognizing the current subtask, transitioning smoothly to the next without explicit stop and re-initialization, and recovering safely after unsuccessful trials. This will likely require a hierarchical framework [186, 187, 188, 189], where TCE and TOP-ERL operate as low-level policies that predict trajectory segments, while a high-level manager selects and sequences these segments to achieve long-term goals. Such an approach would also benefit from incorporating prior knowledge obtained through imitation learning or offline RL, providing the policy with richer contextual understanding and reducing exploration demands. While these extensions present significant technical challenges—including robust long-horizon credit assignment, safety assurance, and scalable task-switching—they represent an important frontier for making ERL applicable to a much broader range of real-world robotic scenarios.

Bibliography

[1] Stefan Schaal. "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics". In: *Adaptive motion of animals and machines*. Springer, 2006, pp. 261–280.

[2] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. "Dynamical movement primitives: learning attractor models for motor behaviors". In: *Neural computation* 25.2 (2013), pp. 328–373.

[3] Alexandros Paraschos, Christian Daniel, Jan Peters, and Gerhard Neumann. "Probabilistic movement primitives". In: *Advances in neural information processing systems* 26 (2013).

[4] Muhammet Yunus Seker, Mert Imre, Justus H Piater, and Emre Ugur. "Conditional Neural Movement Primitives." In: *Robotics: Science and Systems*. Vol. 10. 2019.

[5] Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. "Learning to select and generalize striking movements in robot table tennis". In: *The International Journal of Robotics Research* 32.3 (2013), pp. 263–279.

[6] Sebastian Gomez-Gonzalez, Gerhard Neumann, Bernhard Schölkopf, and Jan Peters. "Using probabilistic movement primitives for striking movements". In: *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. IEEE. 2016, pp. 502–508.

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

[8] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[9] Jan Peters and Stefan Schaal. "Reinforcement learning of motor skills with policy gradients". In: *Neural networks* 21.4 (2008), pp. 682–697.

[10] Fabian Otto, Onur Celik, Hongyi Zhou, Hanna Ziesche, Vien Anh Ngo, and Gerhard Neumann. "Deep black-box reinforcement learning with movement primitives". In: *Conference on Robot Learning*. PMLR. 2022, pp. 1244–1265.

[11] A Vaswani. "Attention is all you need". In: *Advances in Neural Information Processing Systems* (2017).

[12] Ge Li, Zeqi Jin, Michael Volpp, Fabian Otto, Rudolf Lioutikov, and Gerhard Neumann. "ProDMP: A Unified Perspective on Dynamic and Probabilistic Movement Primitives". In: *IEEE Robotics and Automation Letters* 8.4 (2023), pp. 2325–2332.

[13] Ge Li, Hongyi Zhou, Dominik Roth, Serge Thilges, Fabian Otto, Rudolf Lioutikov, and Gerhard Neumann. "Open the Black Box: Step-based Policy Updates for Temporally-Correlated Episodic Reinforcement Learning". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=mnipav175N.

[14] Ge Li, Dong Tian, Hongyi Zhou, Xinkai Jiang, Rudolf Lioutikov, and Gerhard Neumann. "TOP-ERL: Transformer-based Off-Policy Episodic Reinforcement Learning". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=N4NhVN30ph.

[15] Shikhar Bahl, Mustafa Mukadam, Abhinav Gupta, and Deepak Pathak. "Neural dynamic policies for end-to-end sensorimotor learning". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 5058–5069.

[16] You Zhou, Jianfeng Gao, and Tamim Asfour. "Learning via-point movement primitives with inter-and extrapolation capabilities". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 4301–4308.

[17] Michael Volpp, Fabian Flürenbrock, Lukas Grossberger, Christian Daniel, and Gerhard Neumann. "Bayesian Context Aggregation for Neural Processes." In: *ICLR*. 2021.

[18] David S. Broomhead and David Lowe. *Multivariable Functional Interpolation and Adaptive Networks*. Tech. rep. RSRE-MEMO-4148. Royal Signals and Radar Establishment (RSRE), 1988.

[19] Roland L. Hardy. "Multiquadric Equations of Topography and Other Irregular Surfaces". In: *Journal of Geophysical Research* 76.8 (1971), pp. 1905–1915. DOI: 10.1029/JB076i008p01905.

[20] Richard von Mises. "Über die "Ganzzahligkeit" der Atomgewichte und verwandte Fragen". In: *Physikalische Zeitschrift* 19 (1918), pp. 490–500.

[21] Apostolos P. Georgopoulos, Andrew B. Schwartz, and Ronald E. Kettner. "Neuronal Population Coding of Movement Direction". In: *Science* 233.4771 (1986), pp. 1416–1419. DOI: 10.1126/science.3749885.

[22] Rok Pahič, Barry Ridge, Andrej Gams, Jun Morimoto, and Aleš Ude. "Training of deep neural networks for the generation of dynamic movement primitives". In: *Neural Networks* (2020).

[23] Sylvain Calinon, Zhibin Li, Tohid Alizadeh, Nikos G Tsagarakis, and Darwin G Caldwell. "Statistical dynamical systems for skills acquisition in humanoids". In: *2012 12th IEEE-RAS International Conference on Humanoid Robots (Humanoids 2012)*. IEEE. 2012, pp. 323–329.

[24] Chenguang Yang, Chuize Chen, Wei He, Rongxin Cui, and Zhijun Li. "Robot learning system based on adaptive neural control and dynamic movement primitives". In: *IEEE transactions on neural networks and learning systems* 30.3 (2018), pp. 777–787.

[25] Franziska Meier and Stefan Schaal. "A probabilistic representation for dynamic movement primitives". In: *arXiv preprint arXiv:1612.05932* (2016).

[26] Heni Ben Amor, Gerhard Neumann, Sanket Kamthe, Oliver Kroemer, and Jan Peters. "Interaction primitives for human-robot cooperation tasks". In: *2014 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2014, pp. 2831–2837.

[27] Guilherme Maeda, Marco Ewerton, Rudolf Lioutikov, Heni Ben Amor, Jan Peters, and Gerhard Neumann. "Learning interaction for collaborative tasks with probabilistic movement primitives". In: *2014 IEEE-RAS International Conference on Humanoid Robots*. IEEE. 2014, pp. 527–534.

[28] RB Ashith Shyam, Peter Lightbody, Gautham Das, Pengcheng Liu, Sebastian Gomez-Gonzalez, and Gerhard Neumann. "Improving local trajectory optimisation using probabilistic movement primitives". In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 2666–2671.

[29] Leonel Rozo and Vedant Dave. "Orientation probabilistic movement primitives on riemannian manifolds". In: *Conference on Robot Learning*. PMLR. 2022, pp. 373–383.

[30] Alexandros Paraschos, Elmar Rueckert, Jan Peters, and Gerhard Neumann. "Model-free probabilistic movement primitives for physical interaction". In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2015, pp. 2860–2866.

[31] Alexandros Paraschos, Rudolf Lioutikov, Jan Peters, and Gerhard Neumann. "Probabilistic prioritization of movement primitives". In: *IEEE Robotics and Automation Letters* 2.4 (2017), pp. 2294–2301.

[32] Arthur E. Hoerl and Robert W. Kennard. "Ridge Regression: Biased Estimation for Nonorthogonal Problems". In: *Technometrics* 12.1 (1970), pp. 55–67. DOI: 10.1080/00401706.1970.10488634.

[33] Rok Pahič, Andrej Gams, Aleš Ude, and Jun Morimoto. "Deep encoder-decoder networks for mapping raw images to dynamic movement primitives". In: *2018 IEEE International Conference on Robotics and Automation*. 2018.

[34] Marta Garnelo, Dan Rosenbaum, Christopher Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. "Conditional neural processes". In: *International conference on machine learning*. PMLR. 2018, pp. 1704–1713.

[35] Fares J Abu-Dakka and Ville Kyrki. "Geometry-aware dynamic movement primitives". In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2020, pp. 4421–4426.

[36] Mete Akbulut, Erhan Oztop, Muhammet Yunus Seker, X Hh, Ahmet Tekden, and Emre Ugur. "Acnmp: Skill transfer and task extrapolation through learning from demonstration and reinforcement learning via representation sharing". In: *Conference on Robot Learning*. PMLR. 2021, pp. 1896–1907.

[37] Michael Przystupa, Faezeh Haghverd, Martin Jagersand, and Samuele Tosatto. "Deep Probabilistic Movement Primitives with a Bayesian Aggregator". In: *arXiv preprint arXiv:2307.05141* (2023).

[38] Tilman Daab, Noémie Jaquier, Christian Dreher, Andre Meixner, Franziska Krebs, and Tamim Asfour. "Incremental learning of full-pose via-point movement primitives on Riemannian manifolds". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 2317–2323.

[39] Paul Maria Scheikl, Nicolas Schreiber, Christoph Haas, Niklas Freymuth, Gerhard Neumann, Rudolf Lioutikov, and Franziska Mathis-Ullrich. "Movement primitive diffusion: Learning gentle robotic manipulation of deformable objects". In: *IEEE Robotics and Automation Letters* 9.6 (2024), pp. 5338–5345.

[40] Joao Carvalho, A Le, Piotr Kicki, Dorothea Koert, and Jan Peters. "Motion planning diffusion: Learning and adapting robot motion planning with diffusion models". In: *arXiv preprint arXiv:2412.19948* (2024).

[41] Onur Celik, Aleksandar Taranovic, and Gerhard Neumann. "Acquiring diverse skills using curriculum reinforcement learning with mixture of experts". In: *arXiv preprint arXiv:2403.06966* (2024).

[42] Yi Zhang, Chao Zeng, Jian Zhang, and Chenguang Yang. "Wavelet Movement Primitives: A Unified Framework for Learning Discrete and Rhythmic Movements". In: *IEEE Robotics and Automation Letters* (2025).

[43] Yanlong Huang, Leonel Rozo, Joao Silvério, and Darwin G Caldwell. "Kernelized movement primitives". In: *The International Journal of Robotics Research* 38.7 (2019), pp. 833–852.

[44] R Bellman. "Dynamic programming, Princeton University Press, New Jersey". In: (1957).

[45] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[46] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8 (1992), pp. 229–256.

[47] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. "Policy gradient methods for reinforcement learning with function approximation". In: *Advances in neural information processing systems* 12 (1999).

[48] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. "Variance reduction techniques for gradient estimates in reinforcement learning". In: *Journal of Machine Learning Research* 5.Nov (2004), pp. 1471–1530.

[49] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. "High-dimensional continuous control using generalized advantage estimation". In: *arXiv preprint arXiv:1506.02438* (2015).

[50] Reuven Y. Rubinstein and Dirk P. Kroese. *Simulation and the Monte Carlo Method*. 2nd. Wiley Series in Probability and Statistics. Hoboken, NJ: John Wiley & Sons, 2011. ISBN: 978-0-470-17794-5.

[51] Shun-ichi Amari. "Natural Gradient Works Efficiently in Learning". In: *Neural Computation* 10.2 (1998), pp. 251–276.

[52] Sham M Kakade. "A natural policy gradient". In: *Advances in neural information processing systems* 14 (2001).

[53] Ronald A Fisher. "On the mathematical foundations of theoretical statistics". In: *Philosophical transactions of the Royal Society of London. Series A, containing papers of a mathematical or physical character* 222.594-604 (1922), pp. 309–368.

[54] Solomon Kullback and Richard A Leibler. "On information and sufficiency". In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86.

[55] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. "Trust region policy optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[56] Magnus R Hestenes, Eduard Stiefel, et al. "Methods of conjugate gradients for solving linear systems". In: *Journal of research of the National Bureau of Standards* 49.6 (1952), pp. 409–436.

[57] Larry Armijo. "Minimization of functions having Lipschitz continuous first partial derivatives". In: *Pacific Journal of mathematics* 16.1 (1966), pp. 1–3.

[58] Herbert Robbins and Sutton Monro. "A stochastic approximation method". In: *The annals of mathematical statistics* (1951), pp. 400–407.

[59] Fabian Otto, Philipp Becker, Ngo Anh Vien, Hanna Carolin Ziesche, and Gerhard Neumann. "Differentiable trust region layers for deep reinforcement learning". In: *International Conference on Learning Representations* (2021).

[60] Asuka Takatsu. "Wasserstein geometry of Gaussian measures". In: (2011).

[61] Roger A Horn and Charles R Johnson. *Topics in matrix analysis*. Cambridge university press, 1994.

[62] Abbas Abdolmaleki, Rudolf Lioutikov, Jan R Peters, Nuno Lau, Luis Pualo Reis, and Gerhard Neumann. "Model-based relative entropy stochastic search". In: *Advances in Neural Information Processing Systems* 28 (2015).

[63] Oleg Arenz, Gerhard Neumann, and Mingjun Zhong. "Efficient gradient-free variational inference using policy search". In: *International conference on machine learning*. PMLR. 2018, pp. 234–243.

[64] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[65] Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, Larry Rudolph, and Aleksander Madry. "Implementation matters in deep rl: A case study on ppo and trpo". In: *International conference on learning representations*. 2019.

[66] Christopher John Cornish Hellaby Watkins et al. "Learning from delayed rewards". In: (1989).

[67] Long-Ji Lin. "Self-improving reactive agents based on reinforcement learning, planning and teaching". In: *Machine learning* 8.3 (1992), pp. 293–321.

[68] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. "Human-level control through deep reinforcement learning". In: *nature* 518.7540 (2015), pp. 529–533.

[69] Damien Ernst, Pierre Geurts, and Louis Wehenkel. "Tree-based batch mode reinforcement learning". In: *Journal of Machine Learning Research* 6 (2005).

[70] Arthur L. Samuel. “Some Studies in Machine Learning Using the Game of Checkers”. In: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229. DOI: 10.1147/rd.33.0210.

[71] Richard S. Sutton. “Learning to Predict by the Methods of Temporal Differences”. In: *Machine Learning* 3.1 (1988), pp. 9–44. DOI: 10.1007/BF00115009.

[72] Doina Precup, Richard S. Sutton, and Satinder P. Singh. “Eligibility Traces for Off-Policy Policy Evaluation”. In: *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*. San Francisco, CA: Morgan Kaufmann, 2000, pp. 759–766.

[73] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[74] Aviral Kumar, Aurick Zhou, George Tucker, and Sergey Levine. “Conservative Q-Learning for Offline Reinforcement Learning”. In: *Advances in Neural Information Processing Systems 33*. 2020, pp. 1179–1191.

[75] Long-Ji Lin. “Reinforcement Learning for Robots Using Neural Networks”. Technical Report CMU-CS-93-103. PhD thesis. Carnegie Mellon University, 1993.

[76] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. “Prioritized Experience Replay”. In: *International Conference on Learning Representations (ICLR)*. 2016.

[77] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. “Hindsight experience replay”. In: *Advances in neural information processing systems* 30 (2017).

[78] Christopher J. C. Watkins and Peter Dayan. “Q-Learning”. In: *Machine Learning* 8 (1992), pp. 279–292.

[79] Sebastian Thrun and Anton Schwartz. “Issues in using function approximation for reinforcement learning”. In: *Proceedings of the 1993 connectionist models summer school*. Psychology Press. 2014, pp. 255–263.

[80] Ted Moskovitz, Jack Parker-Holder, Aldo Pacchiano, Michael Arbel, and Michael Jordan. “Tactical optimism and pessimism for deep reinforcement learning”. In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 12849–12863.

[81] Hado van Hasselt. “Double Q-learning”. In: *Advances in Neural Information Processing Systems*. 2010.

[82] Hado van Hasselt, Arthur Guez, and David Silver. “Deep Reinforcement Learning with Double Q-learning”. In: *AAAI* (2016).

[83] Marc G. Bellemare, Will Dabney, and Rémi Munos. “A Distributional Perspective on Reinforcement Learning”. In: *International Conference on Machine Learning*. 2017.

[84] Ilya Kostrikov, Ashvin Nair, and Sergey Levine. “Offline reinforcement learning with implicit q-learning”. In: *arXiv preprint arXiv:2110.06169* (2021).

[85] Doina Precup, Richard S Sutton, and Satinder Singh. “Eligibility traces for off-policy policy evaluation”. In: (2000).

[86] Rémi Munos, Tom Stepleton, Anna Harutyunyan, and Marc G. Bellemare. "Safe and Efficient Off-Policy Reinforcement Learning". In: *Advances in Neural Information Processing Systems 29*. Retrace (λ) algorithm. 2016, pp. 1054–1062.

[87] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yee Whye Teh, Razvan Pascanu, and Max Jaderberg. "IMPALA: Scalable Distributed Deep-RL with Importance Weighted Actor-Learner Architectures". In: *International Conference on Machine Learning (ICML)*. 2018, pp. 1406–1415.

[88] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[89] Scott Fujimoto, Herke Hoof, and David Meger. "Addressing function approximation error in actor-critic methods". In: *International conference on machine learning*. PMLR. 2018, pp. 1587–1596.

[90] Darrell Whitley, Stephen Dominic, Rajarshi Das, and Charles W Anderson. "Genetic reinforcement learning for neurocontrol problems". In: *Machine Learning* 13 (1993), pp. 259–284.

[91] Christian Igel. "Neuroevolution for reinforcement learning using evolution strategies". In: *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.* Vol. 4. IEEE. 2003, pp. 2588–2595.

[92] Faustino Gomez, Jürgen Schmidhuber, Risto Miikkulainen, and Melanie Mitchell. "Accelerated Neural Evolution through Cooperatively Coevolved Synapses." In: *Journal of Machine Learning Research* 9.5 (2008).

[93] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. "Evolution strategies as a scalable alternative to reinforcement learning". In: *arXiv preprint arXiv:1703.03864* (2017).

[94] Horia Mania, Aurelia Guy, and Benjamin Recht. "Simple random search of static linear policies is competitive for reinforcement learning". In: *Advances in Neural Information Processing Systems* 31 (2018).

[95] Kai Ploeger, Michael Lutter, and Jan Peters. "High acceleration reinforcement learning for real-world juggling with binary rewards". In: *Conference on Robot Learning*. PMLR. 2021, pp. 642–653.

[96] Fabian Otto, Hongyi Zhou, Onur Celik, Ge Li, Rudolf Lioutikov, and Gerhard Neumann. "MP3: Movement Primitive-Based (Re-) Planning Policy". In: *arXiv preprint arXiv:2306.12729* (2023).

[97] Abbas Abdolmaleki, Bob Price, Nuno Lau, Luis Paulo Reis, and Gerhard Neumann. "Contextual covariance matrix adaptation evolutionary strategies". In: International Joint Conferences on Artificial Intelligence Organization (IJCAI). 2017.

[98] Onur Celik, Dongzhuoran Zhou, Ge Li, Philipp Becker, and Gerhard Neumann. "Specializing versatile skill libraries using local mixture of experts". In: *Conference on Robot Learning*. PMLR. 2022, pp. 1423–1433.

[99] Piotr Kicki, Davide Tateo, Puze Liu, Jonas Günster, Jan Peters, and Krzysztof Walas. "Bridging the gap between Learning-to-plan, Motion Primitives and Safe Reinforcement Learning". In: *8th Annual Conference on Robot Learning*. 2024. URL: https://openreview.net/forum?id=ZdgaF8fOc0.

[100] Nikolaus Hansen and Andreas Ostermeier. "Completely Derandomized Self-Adaptation in Evolution Strategies". In: *Evolutionary Computation* 9.2 (2001), pp. 159–195. DOI: 10.1162/106365601750190398.

[101] Reuven Y. Rubinstein. "The Cross-Entropy Method for Combinatorial and Continuous Optimization". In: *Methodology and Computing in Applied Probability* 1.2 (1999), pp. 127–190. DOI: 10.1023/A:1010091220143.

[102] Voot Tangkaratt, Herke Van Hoof, Simone Parisi, Gerhard Neumann, Jan Peters, and Masashi Sugiyama. "Policy search with high-dimensional context variables". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 31. 1. 2017.

[103] Richard S Sutton. "Learning to predict by the methods of temporal differences". In: *Machine learning* 3 (1988), pp. 9–44.

[104] André-Louis Cholesky. "Sur la résolution numérique des systèmes d'équations linéaires". In: *Bulletin de la Sabix. Société des amis de la Bibliothèque et de l'Histoire de l'École polytechnique* 39 (2005), pp. 81–95.

[105] José C. Pinheiro and Douglas M. Bates. "Unconstrained Parameterizations for Variance-Covariance Matrices". In: *Statistics and Computing* 6.3 (1996), pp. 289–296.

[106] Charles Dugas, Yoshua Bengio, François Bélisle, Claude Nadeau, and René Garcia. "Incorporating second-order functional knowledge for better option pricing". In: *Advances in neural information processing systems* 13 (2000).

[107] Diederik P. Kingma and Max Welling. "Auto-Encoding Variational Bayes". In: *arXiv preprint arXiv:1312.6114* (2013). URL: https://arxiv.org/abs/1312.6114.

[108] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models". In: *Proceedings of the 31st International Conference on Machine Learning (ICML)*. 2014, pp. 1278–1286. URL: https://arxiv.org/abs/1401.4082.

[109] Ronald J. Williams and Jing Peng. "Function Optimization Using Connectionist Reinforcement Learning Algorithms". In: *Connection Science* 3.3 (1991), pp. 241–268. DOI: 10.1080/09540099108946587.

[110] Maximilian Seitzer, Arash Tavakoli, Dimitrije Antic, and Georg Martius. "On the pitfalls of heteroscedastic uncertainty estimation with probabilistic neural networks". In: *arXiv preprint arXiv:2203.09168* (2022).

[111] Marta Garnelo, Dan Rosenbaum, Chris Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo Rezende, and SM Ali Eslami. "Neural Processes". In: *Proceedings of the 35th International Conference on Machine Learning (ICML)*. 2018.

[112] Geoffrey E Hinton. "Training Products of Experts by Minimizing Contrastive Divergence". In: *Neural Computation* 14.8 (2002), pp. 1771–1800.

[113] Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, SM Ali Eslami, Dan Rosenbaum, and Danilo J Rezende. "Attentive Neural Processes". In: *International Conference on Learning Representations (ICLR)*. 2019.

[114] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[115] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1724–1734.

[116] Jacob Devlin. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

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

[118] Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. "Self-attention with relative position representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2018, pp. 464–468.

[119] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. "Transformer-XL: Attentive language models beyond a fixed-length context". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*. 2019, pp. 2978–2988.

[120] Jianlin Su, Yujie Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. "Roformer: Enhanced transformer with rotary position embedding". In: *arXiv preprint arXiv:2104.09864* (2021).

[121] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011, pp. 315–323.

[122] Dan Hendrycks and Kevin Gimpel. "Gaussian Error Linear Units (GELUs)". In: *arXiv preprint arXiv:1606.08415* (2016).

[123] Jimmy Lei Ba. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[124] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep Residual Learning for Image Recognition". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778.

[125] Ruibin Xiong, Yi Yang, Di He, Kai Zheng, Shuxin Zhang, Yingce Lan, Liwei Wang, and Tie-Yan Liu. "On Layer Normalization in the Transformer Architecture". In: *Proceedings of the 37th International Conference on Machine Learning (ICML)*. PMLR. 2020, pp. 10524–10533.

[126] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[127] Hongyi Zhang, Yann N Dauphin, and Tengyu Ma. "Fixup Initialization: Residual Learning Without Normalization". In: *International Conference on Learning Representations (ICLR)*. 2019. URL: https://openreview.net/forum?id=H1gsz30cKX.

[128] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. "Generating long sequences with sparse transformers". In: *arXiv preprint arXiv:1904.10509*. 2019.

[129] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. "Transformers are RNNs: Fast autoregressive transformers with linear attention". In: *International Conference on Machine Learning*. 2020, pp. 5156–5165.

[130] Edward Hu, Yelong Shen, Phil Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Lu Wang, and Weizhu Chen. "LoRA: Low-Rank Adaptation of Large Language Models". In: *arXiv preprint arXiv:2106.09685* (2021).

[131] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations (ICLR)*. 2021.

[132] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. "Decision transformer: Reinforcement learning via sequence modeling". In: *Advances in neural information processing systems* 34 (2021), pp. 15084–15097.

[133] Jeffrey L Elman. "Finding structure in time". In: *Cognitive Science* 14.2 (1990), pp. 179–211.

[134] Bram Bakker. "Reinforcement learning with long short-term memory". In: *Advances in neural information processing systems* 14 (2001).

[135] Keiran Paster, Sheila A. McIlraith, and Jimmy Ba. "Planning from Pixels using Inverse Dynamics Models". In: *ArXiv* abs/2012.02419 (2020). URL: https://api.semanticscholar.org/CorpusID:227305788.

[136] Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. "Efficient off-policy meta-reinforcement learning via probabilistic context variables". In: *International conference on machine learning*. PMLR. 2019, pp. 5331–5340.

[137] Lingheng Meng, Rob Gorbet, and Dana Kulić. "Memory-based deep reinforcement learning for pomdps". In: *2021 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2021, pp. 5619–5626.

[138] Luisa M. Zintgraf, Kyriacos Shiarlis, Maximilian Igl, Sebastian Schulze, Yarin Gal, Katja Hofmann, and Shimon Whiteson. "VariBAD: A Very Good Method for Bayes-Adaptive Deep RL via Meta-Learning". In: *ArXiv* abs/1910.08348 (2019). URL: https://api.semanticscholar.org/CorpusID:204788663.

[139] Emilio Parisotto, Francis Song, Jack Rae, Razvan Pascanu, Caglar Gulcehre, Siddhant Jayakumar, Max Jaderberg, Raphael Lopez Kaufman, Aidan Clark, Seb Noury, et al. "Stabilizing transformers for reinforcement learning". In: *International conference on machine learning*. PMLR. 2020, pp. 7487–7498.

[140] Yueh-Hua Wu, Xiaolong Wang, and Masashi Hamaya. "Elastic decision transformer". In: *Advances in Neural Information Processing Systems* 36 (2024).

[141] Taku Yamagata, Ahmed Khalil, and Raul Santos-Rodriguez. "Q-learning decision transformer: Leveraging dynamic programming for conditional sequence modelling in offline rl". In: *International Conference on Machine Learning*. PMLR. 2023, pp. 38989–39007.

[142] Toshihiro Ota. "Decision mamba: Reinforcement learning via sequence modeling with selective state spaces". In: *arXiv preprint arXiv:2403.19925* (2024).

[143] Qinqing Zheng, Amy Zhang, and Aditya Grover. "Online decision transformer". In: *international conference on machine learning*. PMLR. 2022, pp. 27042–27059.

[144] Guilherme Maeda, Marco Ewerton, Gerhard Neumann, Rudolf Lioutikov, and Jan Peters. "Phase estimation for fast action recognition and trajectory generation in human–robot collaboration". In: *The International Journal of Robotics Research* (2017).

[145] Alexandros Paraschos, Christian Daniel, Jan Peters, and Gerhard Neumann. "Using probabilistic movement primitives in robotics". In: *Autonomous Robots* (2018).

[146] Florian Brandherm, Jan Peters, Gerhard Neumann, and Riad Akrour. "Learning replanning policies with direct policy search". In: *IEEE Robotics and Automation Letters* 4.2 (2019), pp. 2196–2203.

[147] Matteo Saveriano, Fares J Abu-Dakka, Aljaz Kramberger, and Luka Peternel. "Dynamic movement primitives in robotics: A tutorial survey". In: *arXiv preprint arXiv:2102.03861* (2021).

[148] Julen Urain, Michele Ginesi, Davide Tateo, and Jan Peters. "Imitationflow: Learning deep stable stochastic dynamic systems by normalizing flows". In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[149] S Mohammad Khansari-Zadeh and Aude Billard. "Learning control Lyapunov function to ensure stability of dynamical system-based robot reaching motions". In: *Robotics and Autonomous Systems* (2014).

[150] William E Boyce, Richard C DiPrima, and Douglas B Meade. *Elementary differential equations and boundary value problems*. John Wiley & Sons, 2021.

[151] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. "Deep sets". In: *Advances in neural information processing systems* 30 (2017).

[152] Emanuel Todorov, Tom Erez, and Yuval Tassa. "Mujoco: A physics engine for model-based control". In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*.

[153] RM Salinas and B Magyar. *ArUco ROS library*. 2015.

[154] Antonin Raffin, Jens Kober, and Freek Stulp. "Smooth exploration for robotic reinforcement learning". In: *Conference on Robot Learning*. PMLR. 2022, pp. 1634–1644.

[155] Pierre Schumacher, Daniel F.B. Haeufle, Dieter Büchler, Syn Schmitt, and Georg Martius. "DEP-RL: Embodied Exploration for Reinforcement Learning in Overactuated and Musculoskeletal Systems". In: *Proceedings of the Eleventh International Conference on Learning Representations (ICLR)*. May 2023. URL: https://openreview.net/forum?id=C-xa_D3oTj6.

[156] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. "Deep Reinforcement Learning at the Edge of the Statistical Precipice". In: *Advances in Neural Information Processing Systems* (2021).

[157] Alex Braylan, Mark Hollenbeck, Elliot Meyerson, and Risto Miikkulainen. "Frame skip is a powerful parameter for learning to play atari". In: *Workshops at the twenty-ninth AAAI conference on artificial intelligence*. 2015.

[158] André Biedenkapp, Raghu Rajan, Frank Hutter, and Marius Lindauer. "TempoRL: Learning when to act". In: *International Conference on Machine Learning*. PMLR. 2021, pp. 914–924.

[159] Haonan Yu, Wei Xu, and Haichao Zhang. "Taac: Temporally abstract actor-critic for continuous control". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 29021–29033.

[160] Thomas Rückstieß, Martin Felder, and Jürgen Schmidhuber. "State-Dependent Exploration for Policy Gradient Methods". In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Walter Daelemans, Bart Goethals, and Katharina Morik. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 234–249. ISBN: 978-3-540-87481-2.

[161] Alberto Silvio Chiappa, Alessandro Marin Vargas, Ann Zixiang Huang, and Alexander Mathis. "Latent exploration for reinforcement learning". In: *Advances in Neural Information Processing Systems (NeurIPS)* (2023).

[162] Onno Eberhard, Jakob Hollenstein, Cristina Pinneri, and Georg Martius. "Pink noise is all you need: Colored noise exploration in deep reinforcement learning". In: *The Eleventh International Conference on Learning Representations*. 2022.

[163] Jens Kober and Jan Peters. "Policy search for motor primitives in robotics". In: *Advances in neural information processing systems* 21 (2008).

[164] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. "Differentiable convex optimization layers". In: *Advances in neural information processing systems* 32 (2019).

[165] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Karol Hausman, Chelsea Finn, and Sergey Levine. "Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning". In: *Conference on robot learning*. PMLR. 2020, pp. 1094–1100.

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

[167] Michael Wininger, Nam-Hun Kim, and William Craelius. "Spatial resolution of spontaneous accelerations in reaching tasks". In: *Journal of Biomechanics* 42.1 (2009), pp. 29–34.

[168] Neville Hogan and Dagmar Sternad. "Sensitivity of smoothness measures to movement duration, amplitude, and arrests". In: *Journal of motor behavior* 41.6 (2009), pp. 529–534.

[169] Lars Berscheid and Torsten Kröger. "Jerk-limited real-time trajectory generation with arbitrary target states". In: *arXiv preprint arXiv:2105.04830* (2021).

[170] Friedrich Lange and Michael Suppa. "Trajectory generation for immediate path-accurate jerk-limited stopping of industrial robots". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 2021–2026.

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

[172] Moritz Reuss, Ömer Erdinç Yağmurlu, Fabian Wenzel, and Rudolf Lioutikov. "Multimodal Diffusion Transformer: Learning Versatile Behavior from Multimodal Goals". In: *Robotics: Science and Systems*. 2024.

[173] Yevgen Chebotar, Quan Vuong, Karol Hausman, Fei Xia, Yao Lu, Alex Irpan, Aviral Kumar, Tianhe Yu, Alexander Herzog, Karl Pertsch, et al. "Q-transformer: Scalable offline reinforcement learning via autoregressive q-functions". In: *Conference on Robot Learning*. PMLR. 2023, pp. 3909–3928.

[174] Xiao Ma and Wu-Jun Li. "Weighting Online Decision Transformer with Episodic Memory for Offline-to-Online Reinforcement Learning". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 10793–10799.

[175] Haichao Zhang, We Xu, and Haonan Yu. "Policy expansion for bridging offline-to-online reinforcement learning". In: *arXiv preprint arXiv:2302.00935* (2023).

[176] Tianwei Ni, Michel Ma, Benjamin Eysenbach, and Pierre-Luc Bacon. "When do transformers shine in rl? decoupling memory from credit assignment". In: *Advances in Neural Information Processing Systems* 36 (2024).

[177] Chenhao Lu, Ruizhe Shi, Yuyao Liu, Kaizhe Hu, Simon Shaolei Du, and Huazhe Xu. "Rethinking Transformers in Solving POMDPs". In: *Forty-first International Conference on Machine Learning*. 2024. URL: https://openreview.net/forum?id=SyY7ScNpGL.

[178] Haoxin Lin, Yihao Sun, Jiaji Zhang, and Yang Yu. "Model-based reinforcement learning with multi-step plan value estimation". In: *ECAI 2023*. IOS Press, 2023, pp. 1481–1488.

[179] Weilin Yuan, Jiaxing Chen, Shaofei Chen, Dawei Feng, Zhenzhen Hu, Peng Li, and Weiwei Zhao. "Transformer in reinforcement learning for decision-making: a survey". In: *Frontiers of Information Technology & Electronic Engineering* 25.6 (2024), pp. 763–790.

[180] Homanga Bharadhwaj, Jay Vakil, Mohit Sharma, Abhinav Gupta, Shubham Tulsiani, and Vikash Kumar. "Roboagent: Generalization and efficiency in robot manipulation via semantic augmentations and action chunking". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 4788–4795.

[181] Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. "RLlib: Abstractions for distributed reinforcement learning". In: *International conference on machine learning*. PMLR. 2018, pp. 3053–3062.

[182] Weiran Liao, Ge Li, Hongyi Zhou, Rudolf Lioutikov, and Gerhard Neumann. "BMP: Bridging the Gap between B-Spline and Movement Primitives". In: *arXiv preprint arXiv:2411.10336* (2024).

[183] Xi Huang, Hongyi Zhou, Ge Li, Yucheng Tang, Weiran Liao, Björn Hein, Tamim Asfour, and Rudolf Lioutikov. "MoRe-ERL: Learning Motion Residuals using Episodic Reinforcement Learning". In: *arXiv preprint arXiv:2508.01409* (2025).

[184] Hongyi Zhou, Weiran Liao, Xi Huang, Yucheng Tang, Fabian Otto, Xiaogang Jia, Xinkai Jiang, Simon Hilber, Ge Li, Qian Wang, et al. "BEAST: Efficient Tokenization of B-Splines Encoded Action Sequences for Imitation Learning". In: *arXiv preprint arXiv:2506.06072* (2025).

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

[186] Richard S. Sutton, Doina Precup, and Satinder Singh. "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning". In: *Artificial Intelligence* 112.1-2 (1999), pp. 181–211. DOI: 10.1016/S0004-3702(99)00052-1.

[187] Ofir Nachum, Shixiang Shane Gu, Honglak Lee, and Sergey Levine. "Data-efficient hierarchical reinforcement learning". In: *Advances in neural information processing systems* 31 (2018).

[188] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. "Learning multi-level hierarchies with hindsight". In: *arXiv preprint arXiv:1712.00948* (2017).

[189] Tejas D. Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Joshua B. Tenenbaum. "Hierarchical Deep Reinforcement Learning: Integrating Temporal Abstraction and Intrinsic Motivation". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016, pp. 3675–3683.

[190] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268 (2021), pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

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

[192] Thomas Rückstiess, Frank Sehnke, Tom Schaul, Daan Wierstra, Yi Sun, and Jürgen Schmidhuber. "Exploring parameter space in reinforcement learning". In: *Paladyn* 1 (2010), pp. 14–24.

[193] Jens Timmer and Michel Koenig. "On generating power law noise." In: *Astronomy and Astrophysics* 300 (1995), p. 707.

[194] Yann LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, 1998, pp. 9–50.

[195] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. "Rectifier nonlinearities improve neural network acoustic models". In: *Proc. ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. Vol. 28. 2013, p. 3.

[196] Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. "The 37 implementation details of proximal policy optimization". In: *The ICLR Blog Track 2023* (2022).

[197] Ilya Loshchilov and Frank Hutter. "Decoupled weight decay regularization". In: *arXiv preprint arXiv:1711.05101* (2017).

[198] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

A. Appendix for Chapter 3

A.1. Experiment Settings and Details

We present our experiment settings and further results in this section to support reproducibility. We will open-source our code and datasets soon.

A.1.1. Predict Digit Writing Trajectory Given One Image

We present the settings of different approaches in Table A.1 and offer a supplementary result of our model with full covariance learning using all types of digits.

Figure A.1.: Supplementary result of one digit's trajectory prediction experiment. Upper row: the original images (0-9). Lower row: sampled trajectories from the predicted ProDMP trajectory distribution using full covariance.

Table A.1.: Experiment settings of writing digit given one original image.

Settings	CNMPs	NN-based DMPs	ProDMPs
Input obs.	One original digit image, size: 40×40		
Predict	Traj. mean and std	DMPs weights + goal	Mean + cov of ProDMPs weights
Form up trajectory	-	Numerical integration	Use linear basis function in Eq.(3.10)
Loss func.	Traj. NLL	Traj. MSE	Traj. PT-loss in Eq. (4.4)
Traj. sampling	From traj. distribution	Unable	First sample weights then use linear basis function Eq.(3.10)
No. of Basis	-	25 per DoF	25 +1 (weights + goal) per DoF
No. of DoFs	2 in task space		
Type of Basis	-	RBF	RBF
Hyper-params.	Optimizer: [Adam, learning rate: 2e-4, weights decay: 5e-5] Dim of latent obs.: 128. DMPs: [α_x: 2, α: 25, τ: 3]		
Obs. encoder Unc. encoder in Fig. 3.4	{ CNN: [kernel: 5, stride: 1, channel in: 1, channel out: 10] Max pooling: [kernel: 2, stride: 2, channel in: 10, channel out: 10] CNN: [kernel: 5, stride: 1, channel in: 10, channel: out: 20, dropout: 0.5] Max pooling: [kernel: 2, stride: 2, channel in: 20, channel out: 20] MLP: [input neurons: 980, hidden: [128, 128, 128], output: 128] }		
Mean. dec. Unc. dec. in Fig. 3.4	ProDMPs Mean/ Std: MLP[input: 128, hidden: [128, 128, 128], output: 54] ProDMPs Cov: MLP[input: 128, hidden: [256, 256, 256, 256], output: 1485] CNMPs Mean/ Std: MLP[input: 128 + 1, hidden: [128, 128, 128], output: 2] NN-DMPs Mean: MLP[input: 128, hidden: [128, 128, 128], output: 54]		

A.1.2. Predict Digit Writing Trajectory Given Three Noisy Images

We present the setting of our experiment in Table A.2, and more sampled trajectories result in Fig. A.2.

Table A.2.: Experiment setting of writing digit given at most 3 noisy images.

Settings	ProDMPs (Cov)
Input observation	One, two, or three noisy digit images generated from the original image
Predict	Mean and covariance of ProDMPs weights
Form up trajectory	Use linear basis function in Eq.(3.10)
Loss func. Traj.	PT-loss in Eq.(4.4)
Traj. sampling	First sample weights, then use linear basis function Eq.(3.10)
No. of Basis and DoFs	25 + 1 per DoF, 2 DoFs in task space, type: RBF
Hyper-params.	Optimizer: [Adam, learning rate: 2e-4, weights decay: 5e-5] Gradient clipping norm: 20. Dim of latent obs.: 128. DMPs: [α_x: 2, α: 25, τ: 3]
Encoders and Decoders	Same as Table A.1

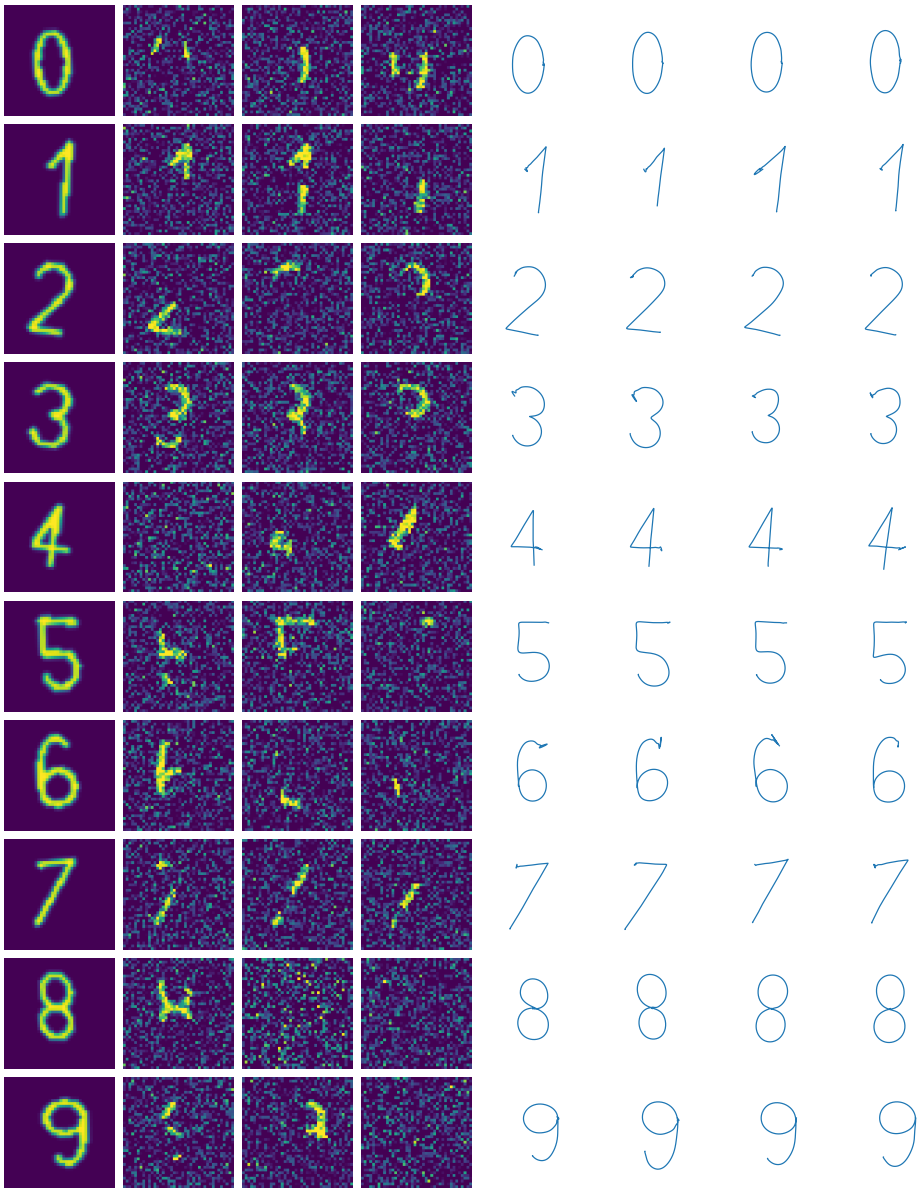


Figure A.2.: More sampled trajectories, from prediction given three noisy images.

A.1.3. Writing a Digit in Three Steps

The setting of our experiment can be found in Table A.3. A supplementary result to Fig. 3.8 can be found in Fig. A.3, where we show examples of all 10 digits' 3-step writing and replanning procedure.

Table A.3.: Experiment setting of writing digit in 3 steps.

Settings	**ProDMPs (Cov) using sub-datasets containing 1 type of digit**
Input observation	One, two, or three noisy digit images generated from the original image
Predict	Mean and covariance of ProDMPs weights
Form up trajectory	Use linear basis function in Eq.(3.10) Use the execution of the previous prediction at 0%, 25% and 50% respectively as boundary condition
Loss func. Traj.	PT-loss in Eq.(4.4)
No. of Basis	25 + 1 per DoF
No. of DoFs	2 in task space
Type of Basis	RBF
Hyper-params.	Optimizer: [Adam, learning rate: 2e-4, weights decay: 5e-5] Gradient clipping norm: 20. Dim of latent obs.: 128. DMPs: [α_x: 2, α: 25, τ: 3]
Encoders	Same as Table A.1
Decoders	Same as Table A.1

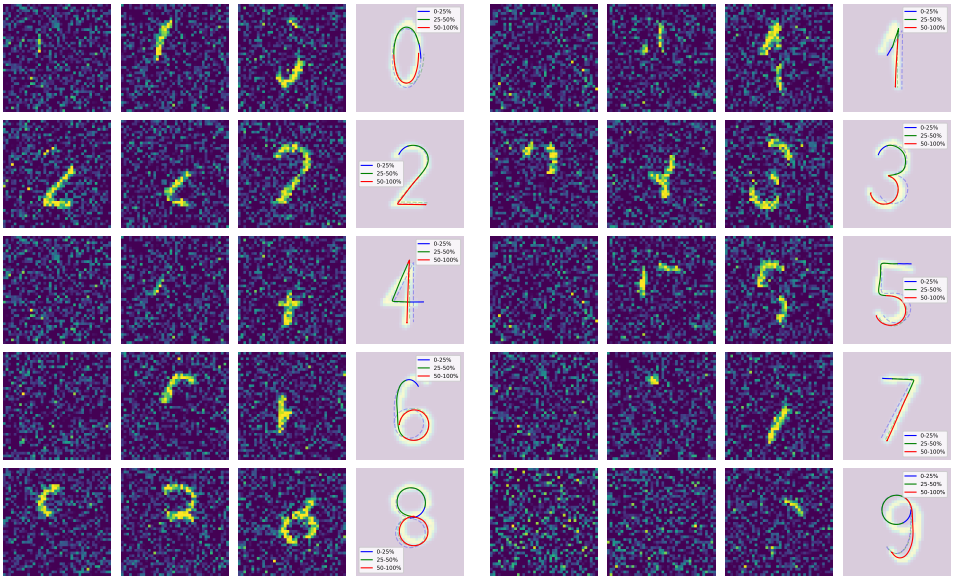


Figure A.3.: Supplementary to Fig 3.8. Here we offer the result of all digits in the 3 steps writing task. We execute the current trajectory while refining it once new information is given.

B. Appendix for Chapter 4

B.1. Experiment Details

B.1.1. Details of Methods Implementation

PPO Proximal Policy Optimization (PPO) [7] is a prominent on-policy step-based RL algorithm that refines the policy gradient objective, ensuring policy updates remain close to the behavior policy. PPO branches into two main variants: PPO-Penalty, which incorporates a KL-divergence term into the objective for regularization, and PPO-Clip, which employs a clipped surrogate objective. In this study, we focus our comparisons on PPO-Clip due to its prevalent use in the field. Our implementation of PPO is based on the implementation of [190].

SAC Soft Actor-Critic (SAC) [8, 191] employs a stochastic step-based policy in an off-policy setting and utilizes double Q-networks to mitigate the overestimation of Q-values for stable updates. By integrating entropy regularization into the learning objective, SAC balances between expected returns and policy entropy, preventing the policy from premature convergence. Our implementation of SAC is based on the implementation of [190].

TRPL Trust Region Projection Layers (TRPL) [59], akin to PPO, addresses the problem of stabilizing the on-policy policy gradient by constraining the learning policy staying close to the behavior policy. TRPL formulates the constrained optimization problem as a projection problem, providing a mathematically rigorous and scalable technique that precisely enforces trust regions on each state, leading to stable and efficient on-policy updates. We evaluated its performance based on the implementation of the original work.

gSDE Generalized State Dependent Exploration (gSDE) [154, 160, 192] is an exploration method designed to address issues with traditional step-based exploration techniques and aims to provide smoother and more efficient exploration in the context of robotic reinforcement learning, reducing jerky motion patterns and potential damage to robot motors while maintaining competitive performance in learning tasks.

To achieve this, gSDE replaces the traditional approach of independently sampling from a Gaussian noise at each time step with a more structured exploration strategy, that samples in a state-dependent manner. The generated samples not only depend on parameter of the Gaussian distribution μ & Σ,

but also on the activations of the policy network's last hidden layer (s). We generate disturbances ϵ_t using the equation

$$\epsilon_t = \theta_\epsilon s, \text{ where } \theta_\epsilon \sim \mathcal{N}^d(0, \Sigma). \tag{B.1}$$

The exploration matrix θ_ϵ is composed of vectors of length $\text{Dim}(a)$ that were drawn from the Gaussian distribution we want gSDE to follow. The vector s describes how this set of pre-computed exploration vectors are mixed. The exploration matrix is resampled at regular intervals, as guided by the 'sde sampling frequency' (ssf), occurring every n-th step if n is our ssf.

gSDE is versatile, applicable as a substitute for the Gaussian Noise source in numerous on- and off-policy algorithms. We evaluated its performance in an on-policy setting using PPO by utilizing the reference implementation for gSDE from Raffin et al. [154]. In order for training with gSDE to remain stable and reach high performance the usage of a linear schedule over the clip range had to be used for some environments.

PINK We utilize SAC to evaluate the effectiveness of pink noise for efficient exploration. Eberhard et al. [162] propose to replace the independent action noise ϵ_t of

$$a_t = \mu_t + \sigma_t \cdot \epsilon_t \tag{B.2}$$

with correlated noise from particular random processes, whose power spectral density follow a power law. In particular, the use of pink noise, with the exponent $\beta = 1$ in $S(f) = |\mathcal{F}[\epsilon](f)|^2 \propto f^{-\beta}$, should be considered [162].

We follow the reference implementation and sample chunks of Gaussian pink noise using the inverse Fast Fourier Transform method proposed by Timmer et al. [193]. These noise variables are used for SAC's exploration but the the actor and critic updates sample the independent action distribution without pink noise. Each action dimension uses an independent noise process which causes temporal correlation within each dimension but not across dimensions. Furthermore, we fix the chunk size and maximum period to 10000 which avoids frequent jumps of chunk borders and increases relative power of low frequencies.

BBRL-Cov/Std Black-Box Reinforcement Learning (BBRL) [10, 96] is a recent developed episodic reinforcement learning method. By utilizing ProMPs [3] as the trajectory generator, BBRL learns a policy that explores at the trajectory level. The method can effectively handle sparse and non-Markovian rewards by perceiving an entire trajectory as a unified data point, neglecting the temporal structure within sampled trajectories. However, on the other hand, BBRL suffers from relatively low sample efficiency due to its black-box nature. Moreover, the original BBRL employs a degenerate Gaussian policy with diagonal covariance. In this study, we extend BBRL to learn Gaussian policy with full covariance to build a more competitive baseline. For clarity, we refer to the original method as BBRL-Std and the full covariance version as BBRL-Cov. We integrate BBRL with ProDMPs [12], aiming to isolate the effects attributable to different MP approaches.

B.1.2. Metaworld Performance Profile Analysis

The distribution of success rates, reported in the performance profile in Fig. 4.5b, may seem to contradict the nearly perfect IQM of TCE but in reality provide insight into the consistency of TCE. Nearly two thirds of runs exceed 99% success rate and are therefore able to perfectly solve the task with this seed. The individual performances reported in Appendix B.1.3 show that only very few tasks, e.g., Disassemble and Hammer, have a significant fraction of unsuccessful seeds. This consistency per task is also visible in the profile, as only two percent of runs fall between zero and sixty percent success rate which is visible by the near zero slope in this range. All methods are entirely unable to solve a small set of tasks and therefore show a gap in the profile. This does not contradict the very high IQM success rate of TCE, as the IQM trims the upper and lower 25% of results. The commonly reported median effectively trims 50% and would result in even higher values. Due to the smaller and later decline in the profile compared to the other methods, the intersection between 75% of runs and the profile is located at a success rate of 90%. Therefore, only a small fraction of runs, roughly ten percent, fall within the 25% trim but only slightly decrease the value of the IQM due their high success rate. Other methods have a larger fraction of imperfect runs with lower success rate within the quartiles.

B.1.3. Performance on Individual Metaworld Tasks

We report the Interquartile Mean (IQM) of success rates for each Metaworld task. The plots clearly illustrate the varying levels of difficulty across different tasks.

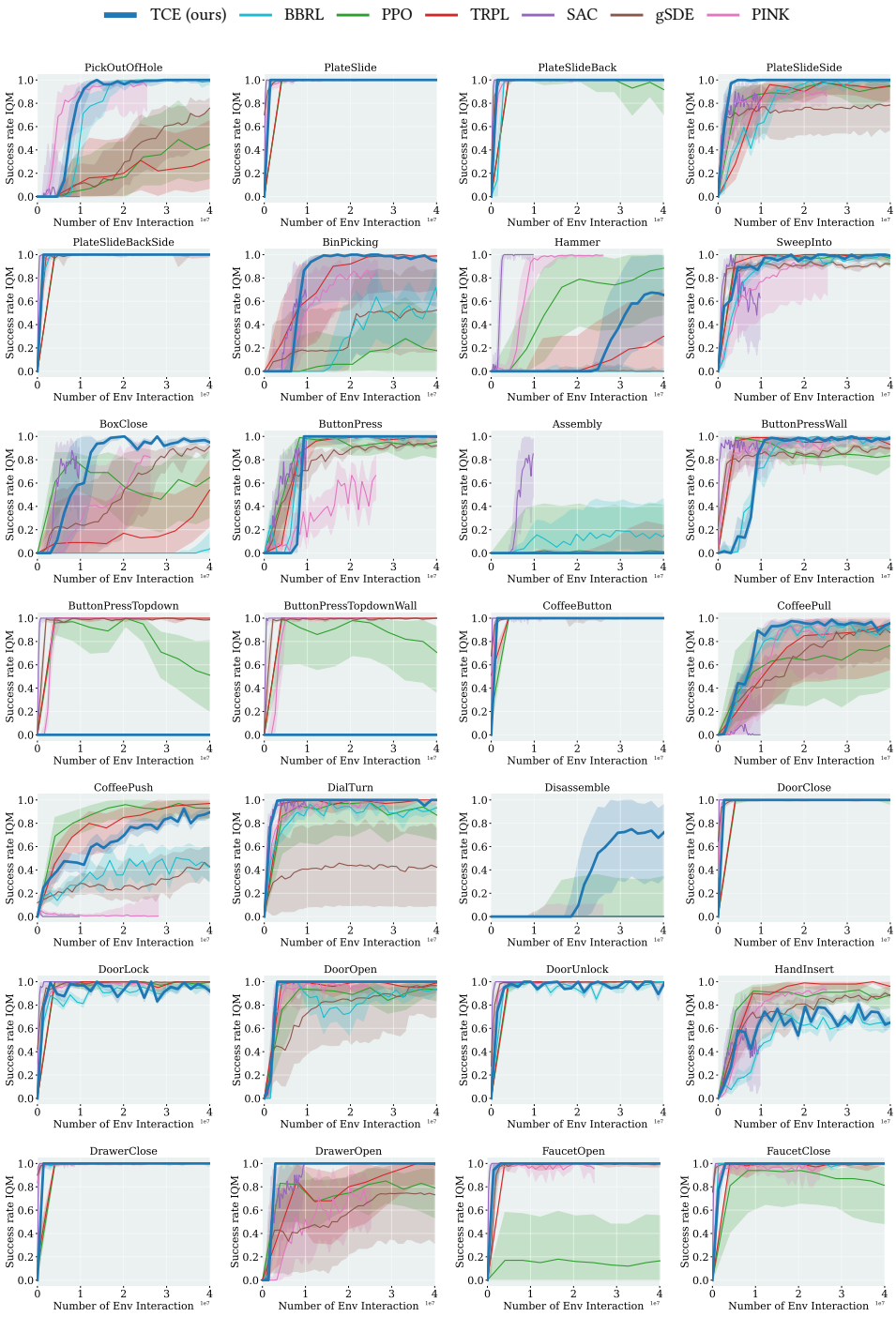


Figure B.1.: Success Rate IQM of each individual Metaworld tasks in Chapter 4, Part 1.

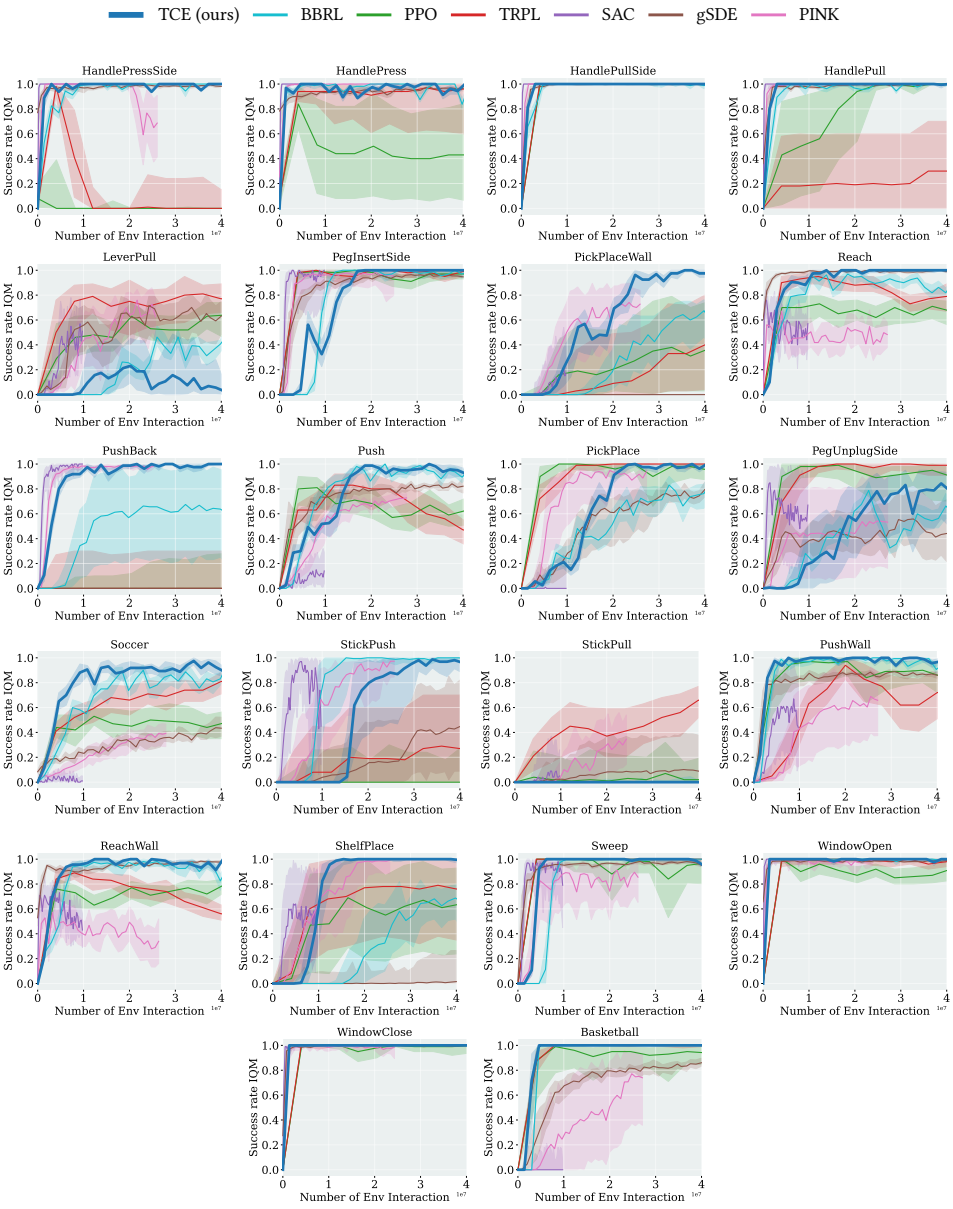
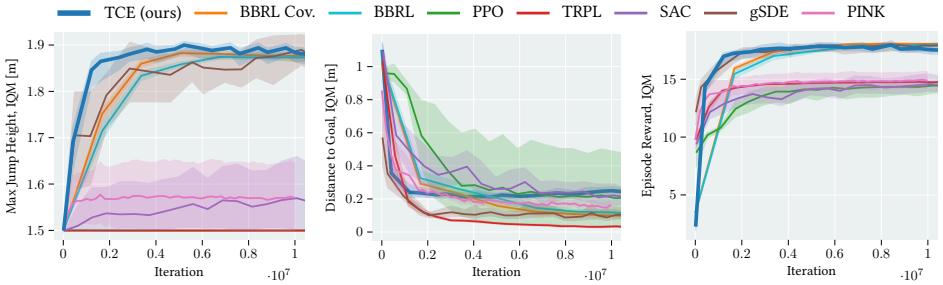


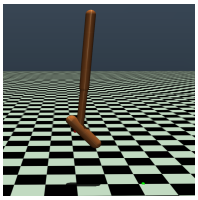
Figure B.2.: Success Rate IQM of each individual Metaworld tasks in Chapter 4, Part 2.

B.1.4. Hopper Jump



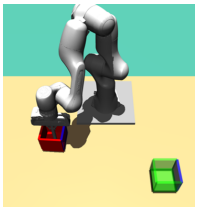
(a) Max Jump Height **(b)** Distance to Goal **(c)** Overall, Height + Dist.

Figure B.3.: Hopper Jump



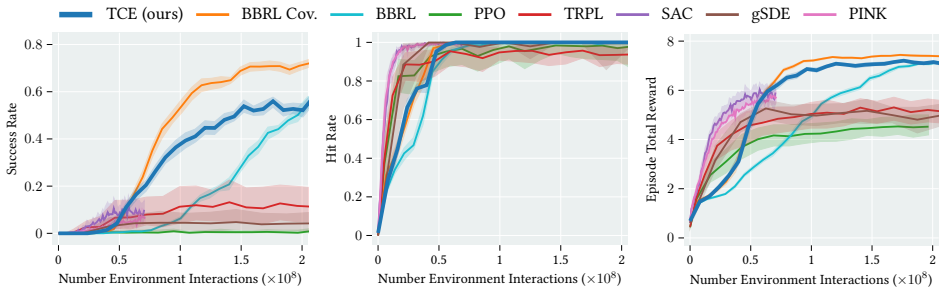
As an addition to the main paper, we provide more details on the Hopper Jump task. We look at both the main goal of maximizing jump height and the secondary goal of landing on a desired position. These are shown along with the overall episode reward in Fig. B.3. Our method shows quick learning and does well in achieving high jump height, consistent with what we reported earlier. While it's not as strong in landing accuracy, it still ranks high in overall performance. Both versions of BBRL have similar results. However, they train more slowly compared to TCE, highlighting the speed advantage of our method due to the use of intermediate states for policy updates. Looking at other methods, step-based ones like PPO and TRPL focus too much on landing distance and miss out on jump height, leading to less effective policies. On the other hand, gSDE performs well but is sensitive to the initial setup, as shown by the wide confidence ranges in the results. Lastly, SAC and PINK shows inconsistent results in jump height, indicating the limitations of using pink noise for exploration, especially when compared to gSDE.

B.1.5. Box Pushing



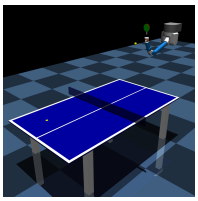
The goal of the box-pushing task is to move a box to a specified goal location and orientation using the 7-DoFs Franka Emika Panda [10]. To make the environment more challenging, we extend the environment from a fixed initial box position and orientation to a randomized initial position and orientation. The range of both initial and target box pose varies from $x \in [0.3, 0.6], y \in [-0.45, 0.45], \theta_z \in [0, 2\pi]$. Success is defined as a positional distance error of less than 5 cm and a z-axis orientation error of less than 0.5 rad. We refer to the original paper for the observation and action spaces definition and the reward function.

B.1.6. Table Tennis



(a) Success Rate **(b)** Hit Rate **(c)** Episode Reward

Figure B.6.: Robot Table Tennis Rand2Rand



The goal of table tennis environment to use the 7-DoFs robotic arm to hit the incoming ball and return it as close as possible to the specified goal location. We adapt the table tennis environment from Celik et al. [98], Otto et al. [10] and extend it to a randomized initial robot joint configuration. As context space we consider the initial ball position $x \in [-1, -0.2]$, $y \in [-0.65, 0.65]$ and the goal position $x \in [-1.2, -0.2]$, $y \in [-0.6, 0.6]$. The task is considered successful if the returned ball lands on the opponent's side of the table and within ≤ 0.2m to the goal location. We refer to the original paper for the observation and action spaces definition and the reward function.

B.1.7. Ablation: SAC + Motion Primitives-based Method

Training movement primitive-based methods using standard RL techniques, such as PPO and SAC, generally poses challenges due to the complex, higher-dimensional trajectory parameter space. In the study by Otto et al. [10], an ablation study employing a PPO-style trust region (likelihood clipping) for training BBRL demonstrated inferior performance compared to the use of a differentiable trust region projection layer.

In Figure B.8, we present an additional ablation study where SAC is used to learn the trajectory parameters of movement primitives. This study compares the performance of SAC with that of the original BBRL and BBRL Cov, leading to relatively poorer performance. The SAC model selected for reporting was the best performer among 40 different combinations of hyperparameters. The hyperparameters adjusted include the output action scaling factor (necessary because the SAC action space is bounded by $[-1, 1]$), policy/critic learning rate, batch size, and the size of the policy/critic network. The relatively shorter training curve of SAC can be attributed to its higher computational cost in policy update [191].

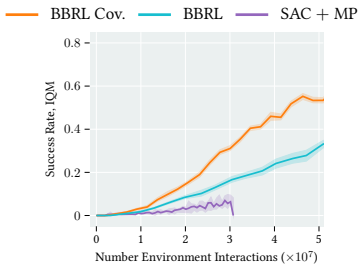


Figure B.8.: By employing the standard SAC method to learn trajectory parameters $\boldsymbol{w}$, we compared its performance with that of the original BBRL and BBRL Cov methods under the box pushing dense reward setting. The ablated method, using SAC, showed relatively poorer performance.

B.1.8. Ablation: Using PPO Style Trust Regions for TCE Method

We developed an ablated version of our method, incorporating the PPO-style trust region via likelihood clipping. We tuned the clipping ratio ϵ between 0.05 and 0.2. As illustrated in Figure B.9, this version's performance falls between the original TCE and the standard PPO. The movement primitives' high-dimensional parameter space limits the effectiveness of likelihood clipping in precisely maintaining the trust region during policy updates. This limitation likely accounts for the performance gap between TCE and its PPO variant. Nonetheless, the ablated model still demonstrates a notable advantage over standard PPO, further substantiating our model's effectiveness in temporally correlated trajectory prediction.

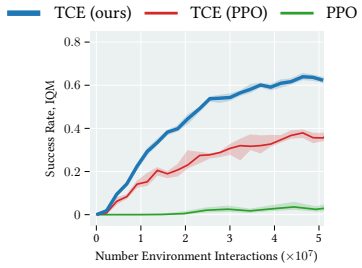
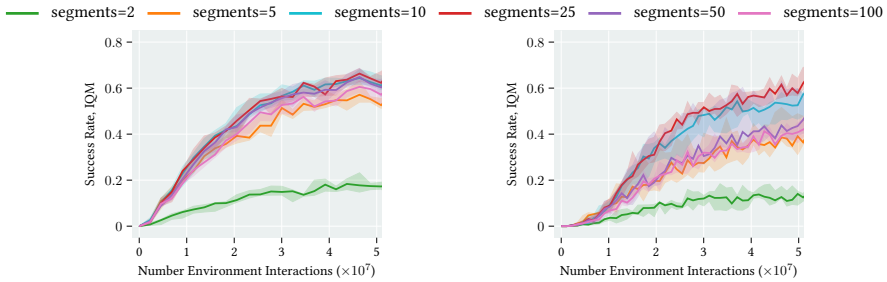


Figure B.9.: Training TCE with a PPO-style trust region, employing likelihood clipping with $\epsilon = 0.1$, yields suboptimal performance in the box pushing dense reward setting. Nevertheless, its superior performance compared to standard PPO underscores our method's effectiveness in episodic trajectory modeling.

B.1.9. Ablation: Selection of the Amount of Segments K

We conducted an ablation study to evaluate the effect of varying the number of segments (k) on model performance. The number of segments tested ranged from 2 to 100. Our experiments involved training in both dense and sparse box-pushing environments. The results revealed a greater sensitivity to the number of segments in the sparse reward environment compared to the dense

environment. We attribute this to the challenges associated with the value function approximation under sparse reward settings. However, within an optimal range, such as 10-25 segments, this parameter is not overly sensitive compared to other hyper-parameters. Consequently, we have adopted k=25 for all experiments in this paper.



(a) Box Pushing, Dense **(b)** Box Pushing, Sparse

Figure B.10.: Study of the number of segment per trajectory.

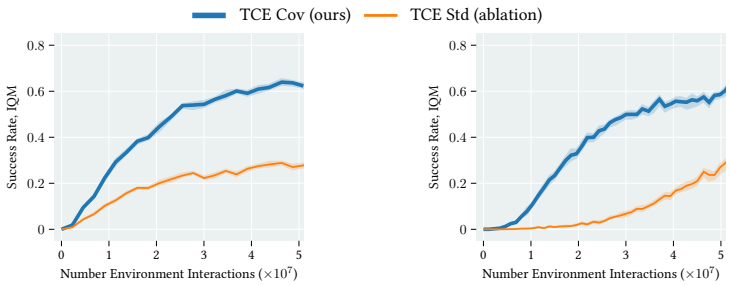
B.1.10. TCE Cov vs. STD

We conducted an ablation study to assess the impact of employing a full covariance policy in the TCE framework. This involved comparing the standard TCE with its variant, TCE Std, which utilizes a factorized Gaussian policy $\mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu_w}, \boldsymbol{\sigma}^2_{\boldsymbol{w}})$. The comparison was conducted in scenarios involving both dense and sparse reward settings in box pushing tasks. The findings revealed that the ablated version, TCE Std, consistently underperformed compared to the full covariance version. This underperformance is attributed to the limited correlation capacity of the factorized Gaussian policy.

Furthermore, it is important to note that while the factorized Gaussian distribution results in a relatively lower computational load in the parameter space, it does not offer a marked advantage when translated into trajectory space. As illustrated in figure 4.7(c) of Section 4.5.4, a factorized parameter distribution ultimately converts into a blocked diagonal trajectory distribution. Although this format is visually simpler compared to a full trajectory covariance matrix, both share same time complexity in terms of likelihood computation. This computational process is significantly more resource-intensive than that for a purely diagonal matrix. Therefore, we utilize the techniques in Li et al. [12] to apply a likelihood estimation and reduce the computational cost.

B.2. Hyperparameters

We executed a large-scale grid search to fine-tune key hyperparameters for each baseline method. For other hyperparameters, we relied on the values specified in their respective original papers. Below is a list summarizing the parameters we swept through during this process.



(a) Box Pushing, Dense

(b) Box Pushing, Sparse

Figure B.11.: Study of the TCE Cov vs. TCE Std.

BBRL: Policy net size, critic net size, policy learning rate, critic learning rate, samples per iteration, trust region dissimilarity bounds, number of parameters per movement DoF.

TCE: Same types of hyper-parameters listed in BBRL, plus the number of segments per trajectory. A learning rate decaying scheduler is applied to stabilize the training in the end.

PPO: Policy network size, critic network size, policy learning rate, critic learning rate, batch size, samples per iteration.

TRPL: Policy network size, critic network size, policy learning rate, critic learning rate, batch size, samples per iteration, trust region dissimilarity bounds.

gSDE: Same types of hyper-parameters listed in PPO, together with the state dependent exploration sampling frequency [154].

SAC: Policy network size, critic network size, policy learning rate, critic learning rate, alpha learning rate, batch size, Update-To-Data (UTD) ratio.

PINK: Same types of hyper-parameters listed in SAC.

The detailed hyperparameters used are listed in the following tables. Unless stated otherwise, the notation lin_x refers to a linear schedule. It interpolates linearly from x to 0 during training. The ERL methods (TCE, BBRL) take an entire trajectory as a sample where the SRL methods take one time step as a sample. In this way, one sample in ERL is equivlent to T sample of SRL, where T is the length of one task episode.

[1] Linear Schedule from 0.3 to 0.01 during the first 25% of the training. Then continued with 0.01.

Table B.1.: HPs of Chapter 5 for the Meta-World experiments. Episode Length $T = 500$

	PPO	gSDE	TRPL	SAC	PINK	TCE	BBRL
number samples	16000	16000	16000	1000	4	16	16
GAE λ	0.95	0.95	0.95	n.a.	n.a.	0.95	n.a.
discount factor	0.99	0.99	0.99	0.99	0.99	1	1
ϵ_μ	n.a.	n.a.	0.005	n.a.	n.a.	0.005	0.005
ϵ_Σ	n.a.	n.a.	0.0005	n.a.	n.a.	0.0005	0.0005
trust region loss coef.	n.a.	n.a.	10	n.a.	n.a.	1	10
optimizer	adam	adam	adam	adam	adam	adam	adam
epochs	10	10	20	1000	1	50	100
learning rate	3e-4	1e-3	5e-5	3e-4	3e-4	3e-4	3e-4
use critic	True	True	True	True	True	True	True
epochs critic	10	10	10	1000	1	50	100
learning rate critic	3e-4	1e-3	3e-4	3e-4	3e-4	3e-4	3e-4
number minibatches	32	n.a.	64	n.a.	n.a.	n.a.	n.a.
batch size	n.a.	500	n.a.	256	512	n.a.	n.a.
buffer size	n.a.	n.a.	n.a.	1e6	2e6	n.a.	n.a.
learning starts	0	0	0	10000	1e5	0	0
polyak_weight	n.a.	n.a.	n.a.	5e-3	5e-3	n.a.	n.a.
SDE sampling frequency	n.a.	4	n.a.	n.a.	n.a.	n.a.	n.a.
entropy coefficient	0	0	0	auto	auto	0	0
normalized observations	True	True	True	False	False	True	False
normalized rewards	True	True	False	False	False	False	False
observation clip	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
reward clip	10.0	10.0	n.a.	n.a.	n.a.	n.a.	n.a.
critic clip	0.2	lin_0.3[1]	n.a.	n.a.	n.a.	n.a.	n.a.
importance ratio clip	0.2	lin_0.3[1]	n.a.	n.a.	n.a.	n.a.	n.a.
hidden layers	[128, 128]	[128, 128]	[128, 128]	[256, 256]	[256, 256]	[128, 128]	[32, 32]
hidden layers critic	[128, 128]	[128, 128]	[128, 128]	[256, 256]	[256, 256]	[128, 128]	[32, 32]
hidden activation	tanh	tanh	tanh	relu	relu	relu	relu
orthogonal initialization	Yes	No	Yes	fanin [194]	fanin	Yes	Yes
initial std	1.0	0.5	1.0	1.0	1.0	1.0	1.0
Movement Primitive (MP) type	n.a.	n.a.	n.a.	n.a.	n.a.	ProDMPs	ProDMPs
number basis functions	n.a.	n.a.	n.a.	n.a.	n.a.	8	5
weight scale	n.a.	n.a.	n.a.	n.a.	n.a.	0.1	0.1
goal scale	n.a.	n.a.	n.a.	n.a.	n.a.	0.1	0.1

Table B.2.: HPs of Chapter 5 for the Box Pushing Dense, Episode Length $T = 100$

	PPO	gSDE	TRPL	SAC	PINK	TCE	BBRL	BBRL Cov.
number samples	48000	80000	48000	8	8	152	152	152
GAE λ	0.95	0.95	0.95	n.a.	n.a.	0.95	n.a.	n.a.
discount factor	1.0	1.0	1.0	0.99	0.99	1.0	1.0	1.0
ϵ_μ	n.a.	n.a.	0.005	n.a.	n.a.	0.05	0.1	0.05
ϵ_Σ	n.a.	n.a.	0.00005	n.a.	n.a.	0.0005	0.00025	0.0005
trust region loss coef.	n.a.	n.a.	10	n.a.	n.a.	1	10	10
optimizer	adam	adam	adam	adam	adam	adam	adam	adam
epochs	10	10	20	1	1	50	20	20
learning rate	5e-5	1e-4	5e-5	3e-4	3e-4	3e-4	3e-4	3e-4
use critic	True	True	True	True	True	True	True	True
epochs critic	10	10	10	1	1	50	10	10
learning rate critic	1e-4	1e-4	1e-4	3e-4	3e-4	1e-3	3e-4	3e-4
number minibatches	40	n.a.	40	n.a.	n.a.	n.a.	n.a.	n.a.
batch size	n.a.	2000	n.a.	512	512	n.a.	n.a.	n.a.
buffer size	n.a.	n.a.	n.a.	2e6	2e6	n.a.	n.a.	n.a.
learning starts	0	0	0	1e5	1e5	0	0	0
polyak_weight	n.a.	n.a.	n.a.	5e-3	5e-3	n.a.	n.a.	n.a.
SDE sampling frequency	n.a.	4	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
entropy coefficient	0	0.01	0	auto	auto	0	0	0
normalized observations	True	True	True	False	False	True	False	False
normalized rewards	True	True	False	False	False	False	False	False
observation clip	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
reward clip	10.0	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
critic clip	0.2	0.2	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
importance ratio clip	0.2	0.2	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
hidden layers	[512, 512]	[256, 256]	[512, 512]	[256, 256]	[256, 256]	[128, 128]	[128, 128]	[128, 128]
hidden layers critic	[512, 512]	[256, 256]	[512, 512]	[256, 256]	[256, 256]	[256, 256]	[256, 256]	[256, 256]
hidden activation	tanh	tanh	tanh	tanh	tanh	leaky_relu [195]	leaky_relu	leaky_relu
orthogonal initialization	Yes	No	Yes	fanin	fanin	Yes	Yes	Yes
initial std	1.0	0.05	1.0	1.0	1.0	1.0	1.0	1.0
MP type	n.a.	n.a.	n.a.	n.a.	n.a.	ProDMPs	ProDMPs	ProDMPs
number basis functions	n.a.	n.a.	n.a.	n.a.	n.a.	8	8	8
weight scale	n.a.	n.a.	n.a.	n.a.	n.a.	0.3	0.3	0.3
goal scale	n.a.	n.a.	n.a.	n.a.	n.a.	0.3	0.3	0.3

Table B.3.: HPs of Chapter 5 for the Box Pushing Sparse, Episode Length $T = 100$

	PPO	gSDE	TRPL	SAC	PINK	TCE	BBRL	BBRL Cov.
number samples	48000	80000	48000	8	8	76	76	76
GAE λ	0.95	0.95	0.95	n.a.	n.a.	0.95	n.a.	n.a.
discount factor	1.0	1.0	1.0	0.99	0.99	1.0	1.0	1.0
ϵ_μ	n.a.	n.a.	0.005	n.a.	n.a.	0.05	0.1	0.05
ϵ_Σ	n.a.	n.a.	0.00005	n.a.	n.a.	0.0005	0.00025	0.001
trust region loss coef.	n.a.	n.a.	10	n.a.	n.a.	1	10	10
optimizer	adam	adam	adam	adam	adam	adam	adam	adam
epochs	10	10	20	1	1	50	20	20
learning rate	5e-4	1e-4	5e-5	3e-4	3e-4	3e-4	3e-4	3e-4
use critic	True	True	True	True	True	True	True	True
epochs critic	10	10	10	1	1	50	10	10
learning rate critic	1e-4	1e-4	1e-4	3e-4	3e-4	3e-4	3e-4	3e-4
number minibatches	40	n.a.	40	n.a.	n.a.	n.a.	n.a.	n.a.
batch size	n.a.	2000	n.a.	512	512	n.a.	n.a.	n.a.
buffer size	n.a.	n.a.	n.a.	2e6	2e6	n.a.	n.a.	n.a.
learning starts	0	0	0	1e5	1e5	0	0	0
polyak_weight	n.a.	n.a.	n.a.	5e-3	5e-3	n.a.	n.a.	n.a.
SDE sampling frequency	n.a.	4	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
entropy coefficient	0	0.01	0	auto	auto	0	0	0
normalized observations	True	True	True	False	False	True	False	False
normalized rewards	True	True	False	False	False	False	False	False
observation clip	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
reward clip	10.0	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
critic clip	0.2	0.2	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
importance ratio clip	0.2	0.2	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
hidden layers	[512, 512]	[256, 256]	[512, 512]	[256, 256]	[256, 256]	[128, 128]	[128, 128]	[128, 128]
hidden layers critic	[512, 512]	[256, 256]	[512, 512]	[256, 256]	[256, 256]	[256, 256]	[256, 256]	[256, 256]
hidden activation	tanh	tanh	tanh	tanh	tanh	leaky_relu	leaky_relu	leaky_relu
orthogonal initialization	Yes	No	Yes	fanin	fanin	Yes	Yes	Yes
initial std	1.0	0.05	1.0	1.0	1.0	1.0	1.0	1.0
MP type	n.a.	n.a.	n.a.	n.a.	n.a.	ProDMPs	ProDMPs	ProDMPs
number basis functions	n.a.	n.a.	n.a.	n.a.	n.a.	8	8	8
weight scale	n.a.	n.a.	n.a.	n.a.	n.a.	0.3	0.3	0.3
goal scale	n.a.	n.a.	n.a.	n.a.	n.a.	0.3	0.3	0.3

Table B.4.: HPs of Chapter 5 for the Hopper Jump, Episode Length $T = 250$

	PPO	gSDE	TRPL	SAC	PINK	TCE	BBRL	BBRL Cov.
number samples	8000	8192	8000	1000	1	64	64	64
GAE λ	0.95	0.99	0.95	n.a.	n.a.	0.95	n.a.	n.a.
discount factor	1.0	0.999	1.0	0.99	0.99	1.0	1.0	1.0
ϵ_μ	n.a.	n.a.	0.05	n.a.	n.a.	0.1	n.a.	0.005
ϵ_Σ	n.a.	n.a.	0.0005	n.a.	n.a.	0.02	n.a.	0.00005
trust region loss coef.	n.a.	n.a.	10	n.a.	n.a.	1	n.a.	10
optimizer	adam	adam	adam	adam	adam	adam	adam	adam
epochs	10	10	20	1000	1	50	100	100
learning rate	3e-4	9.5e-5	3e-4	1e-4	2e-4	1e-4	1e-4	1e-4
use critic	True	True	True	True	True	True	True	True
epochs critic	10	10	10	1000	1	50	100	100
learning rate critic	3e-4	9.5e-5	3e-4	1e-4	2e-4	1e-4	1e-4	1e-4
number minibatches	40	n.a.	40	n.a.	n.a.	n.a.	n.a.	n.a.
batch size	n.a.	128	n.a.	256	256	n.a.	n.a.	n.a.
buffer size	n.a.	n.a.	n.a.	1e6	1e6	n.a.	n.a.	n.a.
learning starts	0	0	0	10000	1e5	0	0	0
polyak_weight	n.a.	n.a.	n.a.	5e-3	5e-3	n.a.	n.a.	n.a.
SDE sampling frequency	n.a.	8	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
entropy coefficient	0	0.0025	0	auto	auto	0	0	0
normalized observations	True	False	True	False	False	True	False	False
normalized rewards	True	False	False	False	False	False	False	False
observation clip	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
reward clip	10.0	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
critic clip	0.2	lin_0.4	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
importance ratio clip	0.2	lin_0.4	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
hidden layers	[32, 32]	[256, 256]	[32, 32]	[256, 256]	[32, 32]	[128, 128]	[32, 32]	[32, 32]
hidden layers critic	[32, 32]	[256, 256]	[32, 32]	[256, 256]	[32, 32]	[128, 128]	[32, 32]	[32, 32]
hidden activation	tanh	tanh	tanh	relu	relu	leaky_relu	tanh	tanh
orthogonal initialization	Yes	No	Yes	fanin	fanin	Yes	Yes	Yes
initial std	1.0	0.1	1.0	1.0	1.0	1.0	1.0	1.0
MP type	n.a.	n.a.	n.a.	n.a.	n.a.	ProDMPs	ProDMPs	ProDMPs
number basis functions	n.a.	n.a.	n.a.	n.a.	n.a.	3	3	3
weight scale	n.a.	n.a.	n.a.	n.a.	n.a.	1	1	1
goal scale	n.a.	n.a.	n.a.	n.a.	n.a.	1	1	1

Table B.5.: HPs of Chapter 5 for the Table Tennis, Episode Length $T = 300$

	PPO	gSDE	TRPL	SAC	PINK	TCE	BBRL	BBRL Cov.
number samples	48000	24000	48000	8	8	76	76	76
GAE λ	0.95	0.95	0.95	n.a.	n.a.	0.95	n.a.	n.a.
discount factor	1.0	1.0	1.0	0.99	0.99	1.0	1.0	1.0
ϵ_μ	n.a.	n.a.	0.005	n.a.	n.a.	0.005	0.004	0.005
ϵ_Σ	n.a.	n.a.	0.0005	n.a.	n.a.	0.00025	0.000025	0.001
trust region loss coef.	n.a.	n.a.	10	n.a.	n.a.	1	25	25
optimizer	adam	adam	adam	adam	adam	adam	adam	adam
epochs	10	10	20	1	1	50	100	100
learning rate	5e-5	1e-4	5e-5	3e-4	3e-4	3e-4	3e-4	3e-4
use critic	True	True	True	True	True	True	True	True
epochs critic	10	10	10	1	1	50	100	100
learning rate critic	1e-4	1e-4	1e-4	3e-4	3e-4	3e-4	3e-4	3e-4
number minibatches	40	n.a.	40	n.a.	n.a.	n.a.	n.a.	n.a.
batch size	n.a.	4000	n.a.	512	512	n.a.	n.a.	n.a.
buffer size	n.a.	n.a.	n.a.	4e6	4e6	n.a.	n.a.	n.a.
learning starts	0	0	0	1e5	1e5	0	0	0
polyak_weight	n.a.	n.a.	n.a.	5e-3	5e-3	n.a.	n.a.	n.a.
SDE sampling frequency	n.a.	8	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
entropy coefficient	0	0	0	auto	auto	0	0	0
normalized observations	True	True	True	False	False	True	False	False
normalized rewards	True	True	False	False	False	False	False	False
observation clip	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
reward clip	10.0	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
critic clip	0.2	0.2	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
importance ratio clip	0.2	0.2	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
hidden layers	[512, 512]	[256, 256]	[256, 256]	[256, 256]	[256, 256]	[256]	[256]	[256]
hidden layers critic	[512, 512]	[256, 256]	[512, 512]	[256, 256]	[256, 256]	[256, 256]	[256]	[256]
hidden activation	tanh	tanh	tanh	tanh	tanh	tanh	tanh	tanh
orthogonal initialization	Yes	Yes	Yes	fanin	fanin	Yes	Yes	Yes
initial std	1.0	0.1	1.0	1.0	1.0	1.0	1.0	1.0
MP type	n.a.	n.a.	n.a.	n.a.	n.a.	ProDMPs	ProDMPs	ProDMPs
number basis functions	n.a.	n.a.	n.a.	n.a.	n.a.	3	3	3
weight scale	n.a.	n.a.	n.a.	n.a.	n.a.	0.7	0.7	0.7
goal scale	n.a.	n.a.	n.a.	n.a.	n.a.	0.1	0.1	0.1

C. Appendix for Chapter 5

C.1. Further Technical Details of TOP-ERL

C.1.1. Policy Training using SAC-style Reparameterization Trick

We utilize the transformer critic to guide the training of our policy, using the reparameterization trick similar to that introduced by SAC [8]. Given a task initial state $\boldsymbol{s}$, the current policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{s}) \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}})$ predicts the Gaussian parameters of the MP's and samples $\tilde{\boldsymbol{w}}$ as follows:

$$\text{Sample MP parameter vector:} \qquad \tilde{\boldsymbol{w}} = \boldsymbol{\mu}_{\boldsymbol{w}} + \boldsymbol{L}_{\boldsymbol{w}}\boldsymbol{\epsilon}, \;\; \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}). \tag{C.1}$$

Here, $\boldsymbol{L}_{\boldsymbol{w}}$ is the Cholesky decomposition of the covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{w}}$, where $\boldsymbol{L}_{\boldsymbol{w}}\boldsymbol{L}_{\boldsymbol{w}}^T = \boldsymbol{\Sigma}_{\boldsymbol{w}}$. This parameterization technique is commonly used for predicting full covariance Gaussian policies. The term $\boldsymbol{\epsilon}$ is a Gaussian white noise vector with the same dimensionality as $\boldsymbol{w}$. Eq. (C.1) represents the full covariance extension of the reparameterization trick typically used in RL, known as $\tilde{a} = \mu_a + \sigma_a\epsilon, \;\; \epsilon \sim \mathcal{N}(0, 1)$.

The sampled $\tilde{\boldsymbol{w}}$ is then used to compute the new trajectory segments. The resulting trajectory segments are computed using the linear basis function expression in Eq. (4) from Section 3.2, where the coefficients c_1 and c_2 are determined by the initial conditions and solved using Eq.3.15 in Chapter 3.3.3:

$$\text{Compute action segment:} \quad \tilde{\boldsymbol{a}}(t) = \boldsymbol{\Phi}(t)^\intercal\tilde{\boldsymbol{w}} + c_1 y_1(t) + c_2 y_2(t) \tag{C.2}$$

Here, $t = 0 : N$ represents the time interval from the beginning to the end of the k-th segment. There are two design choices for the initial conditions: the first is to always use the task initial state $\boldsymbol{s}$ for all segments, while the second is to use the state $\boldsymbol{s}_0^k$ specific to each segment. While the second choice aligns with the techniques described in Section 4.3.1, our empirical results show that the first choice leads to better performance. To the best of our knowledge, we attribute this interesting finding to the following reason. Although the second design choice ensures better consistency in the input to the value function, the per-segment initial conditions were not provided to the policy for action selection, which introduced challenges in updating the policy. We believe this phenomenon is worth further investigation in future research.

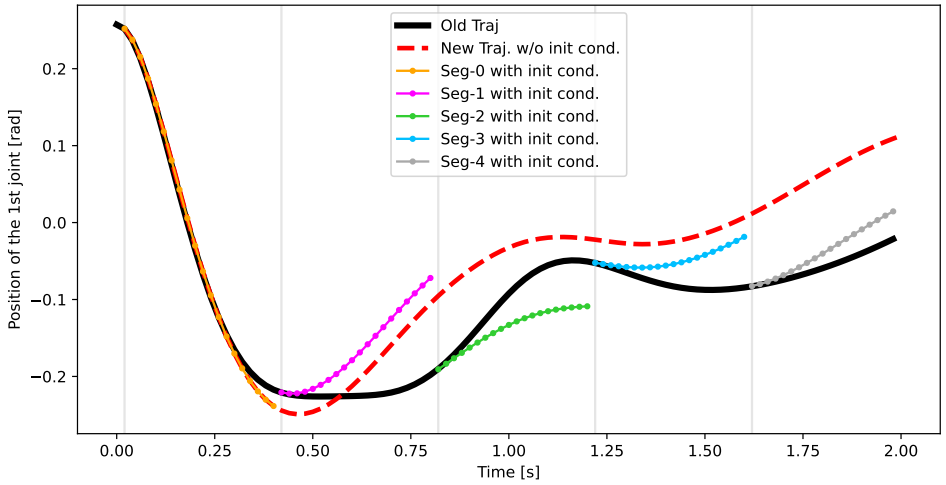


Figure C.1.: TOP-ERL leverages the initial condition enforcement techniques of a dynamic system to ensure that the new action trajectory starts from the corresponding old state. **These action trajectories are taken from the first DoF of the robot in the box pushing task.**

We evaluate and maximize the value of these action segments, using their expectation as the policy's learning objective, as shown in Eq. (9) in Section 4.4:

$$\text{SAC style Objective:} \quad J(\theta) = \mathbb{E}_{\boldsymbol{s}\sim B}\mathbb{E}_{\tilde{\boldsymbol{w}}\sim\pi_\theta(\cdot|\boldsymbol{s})}\left[\frac{1}{KL}\sum_{k=1}^{K}\sum_{N=0}^{L-1} Q_\phi(\boldsymbol{s}_0^k, \left[\tilde{\boldsymbol{a}}_t^k\right]_{t=0:N})\right]. \tag{C.3}$$

Since Eq.(4), (9), (22) and (C.1) are all differentiable, the policy neural network parameters $\boldsymbol{\theta}$ can be trained using gradient ascent. Compared to the technique introduced in SAC, TOP-ERL adds only one additional step: computing the action sequence $[\tilde{\boldsymbol{a}}_t^k]_{t=0:N}$ from the sampled MP parameter $\tilde{\boldsymbol{w}}$.

C.1.2. Enforce trajectory segments' initial condition in TOP-ERL

In Figure C.1, we illustrate how TOP-ERL leverages this mechanism. The figure is based on the motion trajectory of the first degree of freedom of the robot in the box-pushing task. In the critic update, we use five segments as an example.

ProDMP, as a trajectory generator, models the trajectory as a dynamic system. In TOP-ERL, the RL policy predicts ProDMP parameters, which are used to generate a force signal applied to the dynamic system. The system evolves its state based on this force signal and the given initial conditions, such

as the robot's position and velocity at a specific time. The resulting evolution trajectory, shown as the black curve in Fig. C.1, can be computed in closed form and used to control the robot.

When the policy is updated and predicts a new force signal for the same task, a new action trajectory is generated, depicted as the red dashed curve, which gradually deviates from the old trajectory. However, by utilizing the dynamic system's features, we can set the initial condition of each segment of the new trajectory to the corresponding old state. This ensures that the new action sequence used in the target computation in Eq.(7), can start from the old state, as shown across the five segments in the figure. Therefore, we matched the old state and new actions by eliminating the gap between them, as previously discussed in Section 4.3.1.

From our empirical results, we found that this enforcement is highly beneficial for value function learning. Interestingly, for policy updates, it introduces challenges since these conditions are not used as inputs to the policy. Therefore, we apply this technique only during value function training.

C.2. Experiment Details

C.2.1. Details of Methods Implementation

Table C.1.: Baseline methods categorized by type (ERL or SRL) and update rules (On- or Off-policy).

Method	Category	Description
BBRL [10]	ERL, On	Black Box Optimization style ERL, policy search in parameter space
TCE [13]	ERL, On	Extend BBRL to use per-step info for efficient policy update
PPO [7]	SRL, On	Standard on-policy method with simplified Trust Region enforcement
gSDE [154]	SRL, On	Consecutive exploration noise for NN parameters of the policy
GTrXL[139]	SRL, On	Transformer-augmented SRL with multiple state as history
SAC [8]	SRL, Off	Standard off-policy method with entropy bonus for better exploration
PINK [162]	SRL, Off	Use temporal correlated pink noise for better exploration

The description of methods PPO, SAC, PINK, gSDE and BBRL are equivalent to Appendix C.2.1. Method TCE is introduced in Chapter 4.

GTrXL Gated TransformerXL (GTrXL) [139] is a Transformer architecture that design to stabilize the training of Transformers in online RL, offers an easy-to-train, simple-to-implement but substantially more expressive architectural alternative to standard RNNs used for RL agents in POMDPs. Our implementation of GTrXL is based on the implementation of PPO + GTrXL from [181]. We augmented the implementation with minibatch advantage normalization and state-independent log standard deviation as suggested in [196].

C.3. Additional Experiment Results

The environment descriptions are equivalent to Appendix B.1.

We reported each individual Metaworld task in Fig. C.2 and Fig. C.3. These tasks cover a wide range of types and complexities.

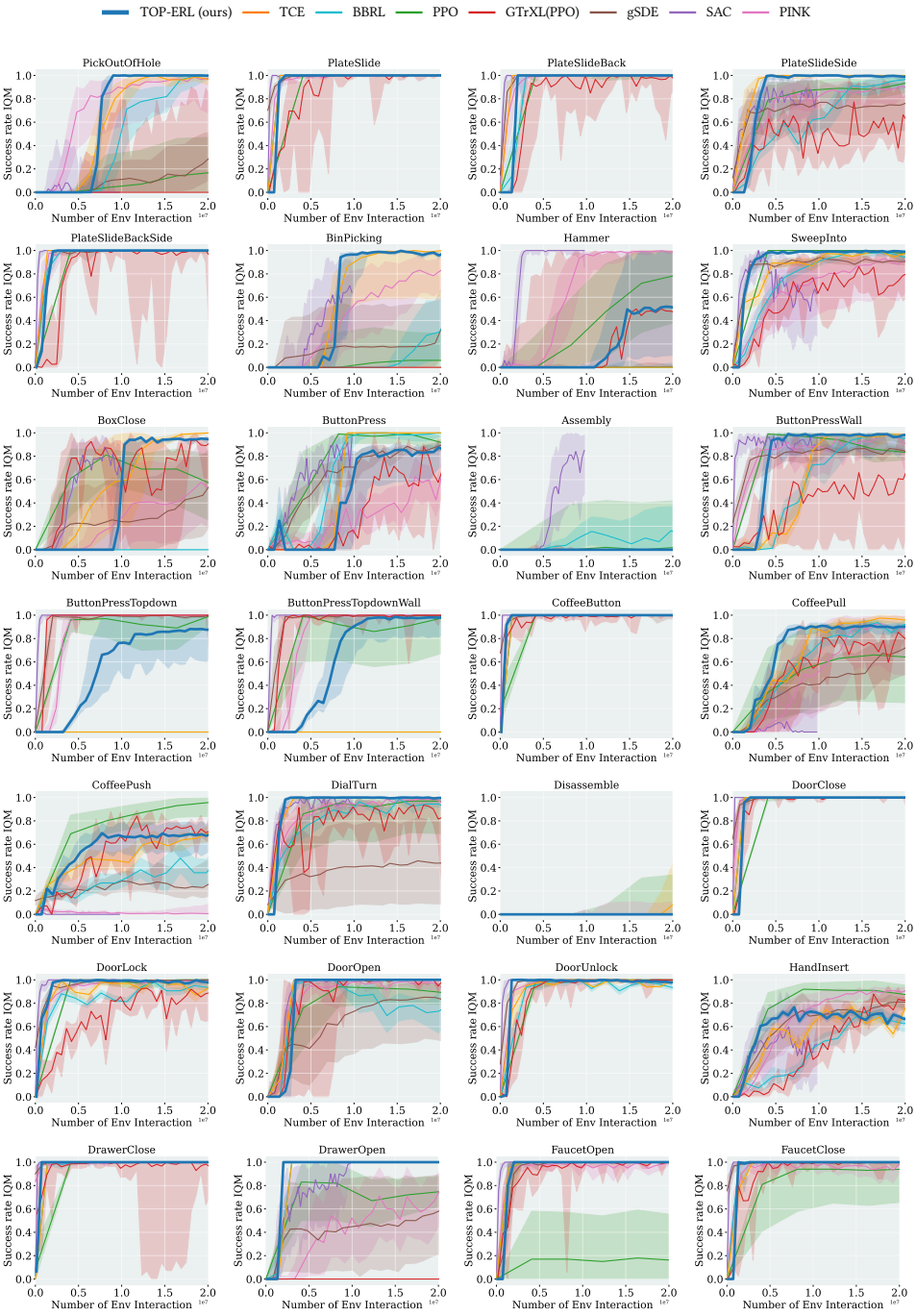


Figure C.2.: Success Rate IQM of each individual Metaworld tasks in Chapter 5, Part 1.

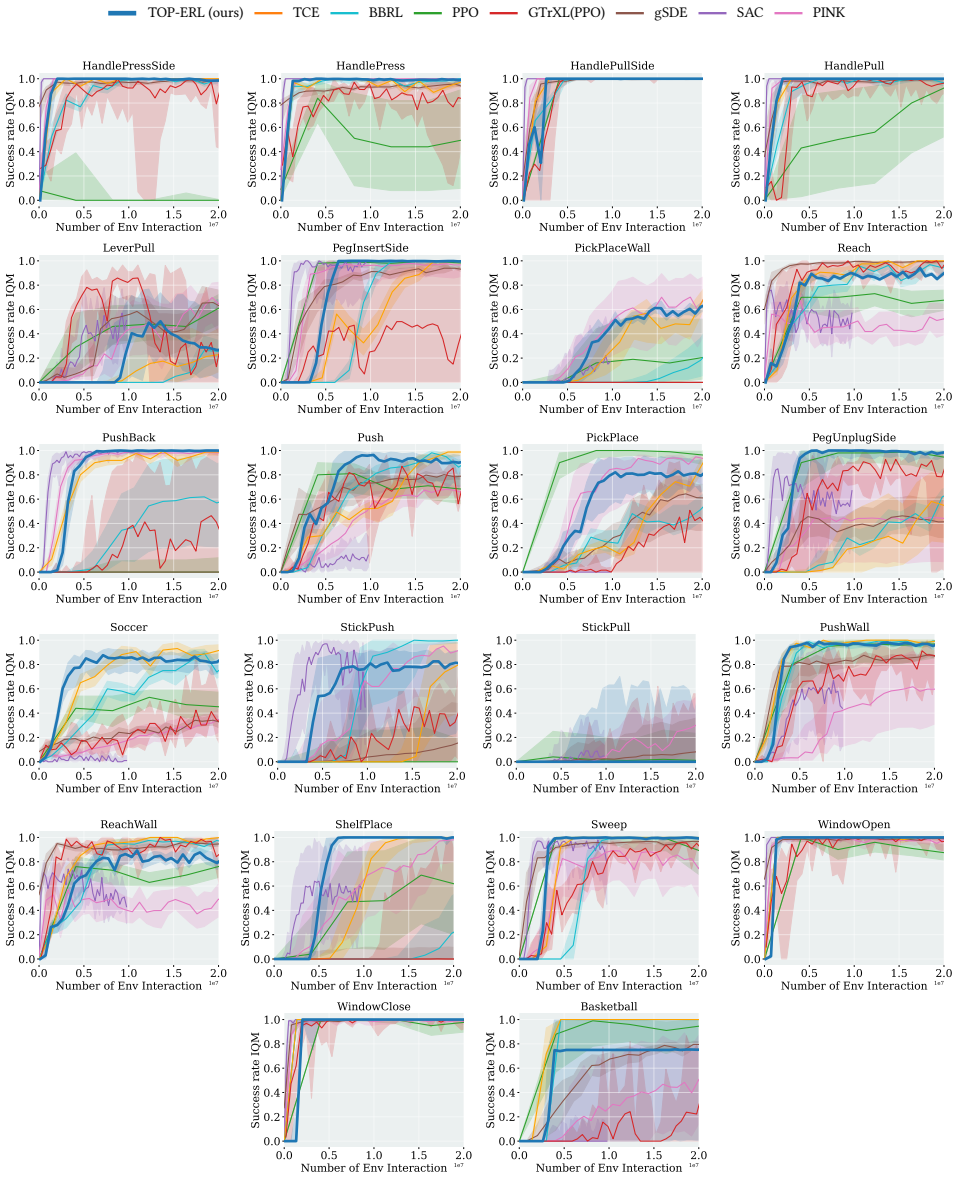


Figure C.3.: Success Rate IQM of each individual Metaworld tasks in Chapter 5, Part 2.

C.4. Hyperparameters for Chapter 5

We executed a large-scale grid search to fine-tune key hyperparameters for each baseline method. For other hyperparameters, we relied on the values specified in their respective original papers. Below is a list summarizing the parameters we swept through during this process.

BBRL: Policy net size, critic net size, policy learning rate, critic learning rate, samples per iteration, trust region dissimilarity bounds, number of parameters per movement DoF.

TCE: Same types of hyper-parameters listed in BBRL, plus the number of segments per trajectory. A learning rate decaying scheduler is applied to stabilize the training in the end.

PPO: Policy network size, critic network size, policy learning rate, critic learning rate, batch size, samples per iteration.

gSDE: Same types of hyper-parameters listed in PPO, together with the state dependent exploration sampling frequency [154].

SAC: Policy network size, critic network size, policy learning rate, critic learning rate, alpha learning rate, batch size, Update-To-Data (UTD) ratio.

PINK: Same types of hyper-parameters listed in SAC.

GTrXL: Number of multi-head attention layers, number of heads, dims per head, importance-sampling ratio clip, value function clip, grad clip, and same hyperparameters listed in PPO

TOP-ERL: Number of multi-head attention layers, number of heads, dims per head, learning rates. The other movement primitives hyper-parameters are taken from TCE.

The detailed hyperparameters used are listed in the following tables. Unless stated otherwise, the notation lin_x refers to a linear schedule. It interpolates linearly from x to 0 during training. The ERL methods (TCE, BBRL) take an entire trajectory as a sample where the SRL methods take one time step as a sample. In this way, one sample in ERL is equivlent to T sample of SRL, where T is the length of one task episode. We choose to use Adam [64] as the neural network optimizer for all models, except for the transformer critic which is updated using AdamW [197]

Table C.2.: HPs of Chapter 6 for the Meta-World experiments. Episode Length $T = 500$

	PPO	gSDE	GTrXL	SAC	PINK	TCE	BBRL	TOP-ERL
number samples	16000	16000	19000	1000	4	16	16	2
GAE λ	0.95	0.95	0.95	n.a.	n.a.	0.95	n.a.	n.a.
discount factor	0.99	0.99	0.99	0.99	0.99	1	1	1.0
ϵ_μ	n.a.	n.a.	n.a.	n.a.	n.a.	0.005	0.005	0.005
ϵ_Σ	n.a.	n.a.	n.a.	n.a.	n.a.	0.0005	0.0005	0.0005
trust region loss coef.	n.a.	n.a.	n.a.	n.a.	n.a.	1	10	1.0
optimizer	adam	adam	adam	adam	adam	adam	adam	adam
epochs	10	10	5	1000	1	50	100	15
learning rate	3e-4	1e-3	2e-4	3e-4	3e-4	3e-4	3e-4	1e-3
use critic	True	True	True	True	True	True	True	True
optimizer critic	adam	adam	adam	adam	adam	adam	adam	adamW
epochs critic	10	10	5	1000	1	50	100	50
learning rate critic	3e-4	1e-3	2e-4	3e-4	3e-4	3e-4	3e-4	5e-5
number minibatches	32	n.a.	n.a	n.a.	n.a.	n.a.	n.a.	n.a.
batch size	n.a.	500	1024	256	512	n.a.	n.a.	256
buffer size	n.a.	n.a.	n.a.	1e6	2e6	n.a.	n.a.	3000
learning starts	0	0	n.a.	10000	1e5	0	0	2
polyak_weight	n.a.	n.a.	n.a.	5e-3	5e-3	n.a.	n.a.	5e-3
SDE sampling frequency	n.a.	4	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
entropy coefficient	0	0	0	auto	auto	0	0	n.a.
normalized observations	True	True	False	False	False	True	False	False
normalized rewards	True	True	0.05	False	False	False	False	False
observation clip	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
reward clip	10.0	10.0	10.0	n.a.	n.a.	n.a.	n.a.	n.a.
critic clip	0.2	lin_0.3	10.0	n.a.	n.a.	n.a.	n.a.	n.a.
importance ratio clip	0.2	lin_0.3	0.1	n.a.	n.a.	n.a.	n.a.	n.a.
hidden layers	[128, 128]	[128, 128]	n.a.	[256, 256]	[256, 256]	[128, 128]	[32, 32]	[128, 128]
hidden layers critic	[128, 128]	[128, 128]	n.a.	[256, 256]	[256, 256]	[128, 128]	[32, 32]	n.a.
hidden activation	tanh	tanh	relu	relu	relu	relu	relu	leaky_relu[195]
orthogonal initialization	Yes	No	xavier [198]	fanin [194]	fanin	Yes	Yes	Yes
initial std	1.0	0.5	1.0	1.0	1.0	1.0	1.0	1.0
number of heads	-	-	4	-	-	-	-	8
dims per head	-	-	16	-	-	-	-	16
number of attention layers	-	-	4	-	-	-	-	2
max sequence length	-	-	5	-	-	-	-	1024

[1] Linear Schedule from 0.3 to 0.01 during the first 25% of the training. Then continued with 0.01.

Table C.3.: HPs of Chapter 6 for the Box Pushing Dense, Episode Length $T = 100$

	PPO	gSDE	GTrXL	SAC	PINK	TCE	BBRL	TOP-ERL
number samples	48000	80000	8000	8	8	152	152	4
GAE λ	0.95	0.95	0.95	n.a.	n.a.	0.95	n.a.	n.a.
discount factor	1.0	1.0	0.99	0.99	0.99	1.0	1.0	1.0
ϵ_μ	n.a.	n.a.	n.a.	n.a.	n.a.	0.05	0.1	0.005
ϵ_Σ	n.a.	n.a.	n.a.	n.a.	n.a.	0.0005	0.00025	0.0005
trust region loss coef.	n.a.	n.a.	n.a.	n.a.	n.a.	1	10	1.0
optimizer	adam	adam	adam	adam	adam	adam	adam	adam
epochs	10	10	5	1	1	50	20	15
learning rate	5e-5	1e-4	2e-4	3e-4	3e-4	3e-4	3e-4	3e-4
use critic	True	True	True	True	True	True	True	True
optimizer critic	adam	adam	adam	adam	adam	adam	adam	adamW
epochs critic	10	10	5	1	1	50	10	30
learning rate critic	1e-4	1e-4	2e-4	3e-4	3e-4	1e-3	3e-4	5e-5
number minibatches	40	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
batch size	n.a.	2000	1000	512	512	n.a.	n.a.	512
buffer size	n.a.	n.a.	n.a.	2e6	2e6	n.a.	n.a.	7000
learning starts	0	0	0	1e5	1e5	0	0	8000
polyak_weight	n.a.	n.a.	n.a.	5e-3	5e-3	n.a.	n.a.	5e-3
SDE sampling frequency	n.a.	4	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
entropy coefficient	0	0.01	0	auto	auto	0	0	0.
normalized observations	True	True	False	False	False	True	False	False
normalized rewards	True	True	0.1	False	False	False	False	False
observation clip	10.0	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
reward clip	10.0	10.0	10.	n.a.	n.a.	n.a.	n.a.	n.a.
critic clip	0.2	0.2	10.	n.a.	n.a.	n.a.	n.a.	n.a.
importance ratio clip	0.2	0.2	0.1	n.a.	n.a.	n.a.	n.a.	n.a.
hidden layers	[512, 512]	[256, 256]	n.a.	[256, 256]	[256, 256]	[128, 128]	[128, 128]	[256, 256]
hidden layers critic	[512, 512]	[256, 256]	n.a.	[256, 256]	[256, 256]	[256, 256]	[256, 256]	n.a.
hidden activation	tanh	tanh	relu	tanh	tanh	leaky_relu	leaky_relu	leaky_relu
orthogonal initialization	Yes	No	xavier	fanin	fanin	Yes	Yes	Yes
initial std	1.0	0.05	1.0	1.0	1.0	1.0	1.0	1.0
number of heads	-	-	4	-	-	-	-	8
dims per head	-	-	16	-	-	-	-	16
number of attention layers	-	-	4	-	-	-	-	2
max sequence length	-	-	5	-	-	-	-	1024
MP type	n.a.	n.a.	value	n.a.	n.a.	ProDMPs	ProDMPs	ProDMPs
number basis functions	n.a.	n.a.	value	n.a.	n.a.	8	8	8
weight scale	n.a.	n.a.	value	n.a.	n.a.	0.3	0.3	0.3
goal scale	n.a.	n.a.	value	n.a.	n.a.	0.3	0.3	0.3

Table C.4.: HPs of Chapter 6 for the Box Pushing Sparse, Episode Length $T = 100$

	PPO	gSDE	GTrXL	SAC	PINK	TCE	BBRL	TOP-ERL
number samples	48000	80000	8000	8	8	76	76	4
GAE λ	0.95	0.95	0.95	n.a.	n.a.	0.95	n.a.	n.a.
discount factor	1.0	1.0	1.0	0.99	0.99	1.0	1.0	1.0
ϵ_μ	n.a.	n.a.	n.a.	n.a.	n.a.	0.05	0.1	0.005
ϵ_Σ	n.a.	n.a.	n.a.	n.a.	n.a.	0.0005	0.00025	0.0005
trust region loss coef.	n.a.	n.a.	n.a.	n.a.	n.a.	1	10	1.0
optimizer	adam	adam	adam	adam	adam	adam	adam	adam
epochs	10	10	5	1	1	50	20	15
learning rate	5e-4	1e-4	2e-4	3e-4	3e-4	3e-4	3e-4	3e-4
use critic	True	True	True	True	True	True	True	True
optimizer critic	adam	adam	adam	adam	adam	adam	adam	adamW
epochs critic	10	10	5	1	1	50	10	30
learning rate critic	1e-4	1e-4	2e-4	3e-4	3e-4	3e-4	3e-4	5e-5
number minibatches	40	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
batch size	n.a.	2000	1000	512	512	n.a.	n.a.	512
buffer size	n.a.	n.a.	n.a.	2e6	2e6	n.a.	n.a.	7000
learning starts	0	0	0	1e5	1e5	0	0	400
polyak_weight	n.a.	n.a.	0	5e-3	5e-3	n.a.	n.a.	5e-3
SDE sampling frequency	n.a.	4	0	n.a.	n.a.	n.a.	n.a.	n.a.
entropy coefficient	0	0.01	0	auto	auto	0	0	0
normalized observations	True	True	False	False	False	True	False	False
normalized rewards	True	True	0.1	False	False	False	False	False
observation clip	10.0	n.a.	False	n.a.	n.a.	n.a.	n.a.	n.a.
reward clip	10.0	10.0	10.0	n.a.	n.a.	n.a.	n.a.	n.a.
critic clip	0.2	0.2	10.0	n.a.	n.a.	n.a.	n.a.	n.a.
importance ratio clip	0.2	0.2	0.1	n.a.	n.a.	n.a.	n.a.	n.a.
hidden layers	[512, 512]	[256, 256]	n.a.	[256, 256]	[256, 256]	[128, 128]	[128, 128]	[256, 256]
hidden layers critic	[512, 512]	[256, 256]	n.a.	[256, 256]	[256, 256]	[256, 256]	[256, 256]	n.a.
hidden activation	tanh	tanh	relu	tanh	tanh	leaky_relu	leaky_relu	leaky_relu
orthogonal initialization	Yes	No	xavier	fanin	fanin	Yes	Yes	Yes
initial std	1.0	0.05	1.0	1.0	1.0	1.0	1.0	1.0
number of heads	-	-	4	-	-	-	-	8
dims per head	-	-	16	-	-	-	-	16
number of attention layers	-	-	4	-	-	-	-	2
max sequence length	-	-	5	-	-	-	-	1024
MP type	n.a.	n.a.	value	n.a.	n.a.	ProDMPs	ProDMPs	ProDMPs
number basis functions	n.a.	n.a.	value	n.a.	n.a.	8	8	8
weight scale	n.a.	n.a.	value	n.a.	n.a.	0.3	0.3	0.3
goal scale	n.a.	n.a.	value	n.a.	n.a.	0.3	0.3	0.3

Table C.5.: HPs of Chapter 6 for the Hopper Jump, Episode Length $T = 250$

	PPO	gSDE	GTrXL	SAC	PINK	TCE	BBRL	TOP-ERL
number samples	8000	8192	10000	1000	1	64	64	1
GAE λ	0.95	0.99	0.95	n.a.	n.a.	0.95	n.a.	n.a.
discount factor	1.0	0.999	1.0	0.99	0.99	1.0	1.0	1.0
ϵ_μ	n.a.	n.a.	n.a.	n.a.	n.a.	0.1	n.a.	0.1
ϵ_Σ	n.a.	n.a.	n.a.	n.a.	n.a.	0.02	n.a.	0.02
trust region loss coef.	n.a.	n.a.	n.a.	n.a.	n.a.	1	n.a.	1.0
optimizer	adam	adam	adam	adam	adam	adam	adam	adam
epochs	10	10	10	1000	1	50	100	10
learning rate	3e-4	9.5e-5	5e-4	1e-4	2e-4	1e-4	1e-4	1e-4
use critic	True	True	True	True	True	True	True	True
optimizer critic	adam	adam	adam	adam	adam	adam	adam	adamW
epochs critic	10	10	10	1000	1	50	100	20
learning rate critic	3e-4	9.5e-5	5e-4	1e-4	2e-4	1e-4	1e-4	5e-5
number minibatches	40	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
batch size	n.a.	128	1024	256	256	n.a.	n.a.	256
buffer size	n.a.	n.a.	n.a.	1e6	1e6	n.a.	n.a.	1000
learning starts	0	0	0	10000	1e5	0	0	250
polyak_weight	n.a.	n.a.	n.a.	5e-3	5e-3	n.a.	n.a.	5e-3
SDE sampling frequency	n.a.	8	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
entropy coefficient	0	0.0025	0.	auto	auto	0	0	0
normalized observations	True	False	False	False	False	True	False	False
normalized rewards	True	False	False	False	False	False	False	False
observation clip	10.0	n.a.	False	n.a.	n.a.	n.a.	n.a.	n.a.
reward clip	10.0	10.0	10.	n.a.	n.a.	n.a.	n.a.	n.a.
critic clip	0.2	lin_0.4	1.	n.a.	n.a.	n.a.	n.a.	n.a.
importance ratio clip	0.2	lin_0.4	0.2	n.a.	n.a.	n.a.	n.a.	n.a.
hidden layers	[32, 32]	[256, 256]	n.a.	[256, 256]	[32, 32]	[128, 128]	[32, 32]	[128, 128]
hidden layers critic	[32, 32]	[256, 256]	n.a	[256, 256]	[32, 32]	[128, 128]	[32, 32]	n.a.
hidden activation	tanh	tanh	relu	relu	relu	leaky_relu	tanh	leaky_relu
orthogonal initialization	Yes	No	xavier	fanin	fanin	Yes	Yes	Yes
initial std	1.0	0.1	1.0	1.0	1.0	1.0	1.0	1.0
number of heads	-	-	4	-	-	-	-	8
dims per head	-	-	16	-	-	-	-	16
number of attention layers	-	-	4	-	-	-	-	2
max sequence length	-	-	5	-	-	-	-	1024
MP type	n.a.	n.a.	value	n.a.	n.a.	ProDMPs	ProDMPs	ProDMPs
number basis functions	n.a.	n.a.	value	n.a.	n.a.	3	3	3
weight scale	n.a.	n.a.	value	n.a.	n.a.	1	1	1
goal scale	n.a.	n.a.	value	n.a.	n.a.	1	1	1

List of Figures

List of Tables

List of Algorithms

Curriculum Vitae

Experience

- **Doctoral Researcher**, Karlsruhe Institute of Technology (KIT), Autonomous Learning Robots Lab, Karlsruhe, Germany, Since March 2020
- **Teaching Assistant**, Machine Learning (WS21/22, SS23, SS24), Cognitive System (SS20, SS21).
- **Research Intern**, Max Planck Institute for Intelligent Systems (MPI-IS), Department of Empirical Inference, Tübingen, Germany, Nov 2018 – July 2019
- **Software Engineer Intern**, Alfred Kärcher SE & Co. KG., Department of Robotics, Winnenden, Germany, March 2018 – Aug 2018

Education

- **Master of Science**, RWTH Aachen University, Computer-aided Conception and Production of Mechanical Engineering, Aachen, Germany, Oct 2015 – July 2019
- **Bachelor of Engineering**, University of Science and Technology of China, Thermal Energy and Power Engineering, Hefei, China, Sept 2011 – July 2015

Reviewer (Machine Learning and Robotics)

- Advances in Neural Information Processing Systems (NeurIPS)
- International Conference on Learning Representations (ICLR)
- Reinforcement Learning Conference (RLC)
- Conference on Robot Learning (CoRL)
- IEEE Robotics and Automation Letters (RA-L)
- IEEE International Conference on Robotics and Automation (ICRA)
- IEEE International Conference on Intelligent Robots and Systems (IROS)

Own Publications

This section contains a complete list of the publications that I have contributed to during my PhD studies.

[1] **Ge Li**, Dong Tian, Hongyi Zhou, Xinkai Jiang, Rudolf Lioutikov, Gerhard Neumann. "TOP-ERL: Transformer-based Off-Policy Episodic Reinforcement Learning". *In: International Conference on Learning Representations (ICLR),* 2025, Spotlight (Top 5%).

[2] **Ge Li**, Hongyi Zhou, Dominik Roth, Serge Thilges, Fabian Otto, Rudolf Lioutikov, and Gerhard Neumann. "Open the Black Box: Step-based Policy Updates for Temporally-Correlated Episodic Reinforcement Learning." *In: International Conference on Learning Representations (ICLR),* 2024.

[3] **Ge Li**, Zeqi Jin, Michael Volpp, Fabian Otto, Rudolf Lioutikov, and Gerhard Neumann. "ProDMP: A Unified Perspective on Dynamic and Probabilistic Movement Primitives." *In: IEEE Robotics and Automation Letters (RAL),* 2023.

[4] Onur Celik, Zechu Li, Denis Blessing, **Ge Li**, Daniel Palenicek, Jan Peters, Georgia Chalvatzaki, Gerhard Neumann, "DIME: Diffusion-Based Maximum Entropy Reinforcement Learning", *In: International Conference on Machine Learning (ICML),* 2025.

[5] Hongyi Zhou, Weiran Liao, Xi Huang, Yucheng Tang, Fabian Otto, Xiaogang Jia, Xinkai Jiang, Simon Hilber, **Ge Li**, Qian Wang, Ömer Erdinç Yağmurlu, Nils Blank, Moritz Reuss, Rudolf Lioutikov, "BEAST: Efficient Tokenization of B-Splines Encoded Action Sequences for Imitation Learning", *In: Advances in Neural Information Processing Systems (NeurIPS),* 2025.

[6] Xiaogang Jia, Qian Wang, Anrui Wang, Han A. Wang, Balázs Gyenes, Emiliyan Gospodinov, Xinkai Jiang, **Ge Li**, Hongyi Zhou, Weiran Liao, Xi Huang, Maximilian Beck, Moritz Reuss, Rudolf Lioutikov, Gerhard Neumann, "PointMapPolicy: Structured Point Cloud Processing for Multi-Modal Imitation Learning", *In: Advances in Neural Information Processing Systems (NeurIPS),* 2025.

[7] Xi Huang, Hongyi Zhou, **Ge Li**, Yucheng Tang, Björn Hein, Tamim Asfour, Rudolf Lioutikov, "MoRe-ERL: Learning Motion Residuals using Episodic Reinforcement Learning", *In: IEEE Robotics and Automation Letters (RAL),* 2025.

[8] Xinkai Jiang, Qihao Yuan, Enes Ulas Dincer, Hongyi Zhou, **Ge Li**, Xueyin Li, Julius Haag, Nicolas Schreiber, Kailai Li, Gerhard Neumann, Rudolf Lioutikov. "IRIS: An Immersive Robot Interaction System". *Conference on Robot Learning (CoRL),* 2025.

[9] Hongyi Zhou, Denis Blessing, **Ge Li**, Onur Celik, Gerhard Neumann, Rudolf Lioutikov, "Variational Distillation of Diffusion Policies into Mixture of Experts", *In: Advances in Neural Information Processing Systems (NeurIPS),* 2024.

[10] Xiaogang Jia, Qian Wang, Atalay Donat, Bowen Xing, **Ge Li**, Hongyi Zhou, Onur Celik, Denis Blessing, Rudolf Lioutikov, Gerhard Neumann, MaIL: Improving Imitation Learning with Selective State Space Models, *In: Conference on Robot Learning (CoRL),* 2024.

[11] Zvezdan Lončarević, **Ge Li**, Gerhard Neumann, Andrej Gams. "An encoder-decoder architecture for smooth motion generation." *In: International Conference on Robotics in Alpe-Adria Danube Region,* 2023.

[12] Onur Celik, Dongzhuoran Zhou, **Ge Li**, Philipp Becker, and Gerhard Neumann. "Specializing versatile skill libraries using local mixture of experts." *In: Conference on Robot Learning (CoRL),* 2021.

[13] Weiran Liao, **Ge Li**, Hongyi Zhou, Rudolf Lioutikov, Gerhard Neumann, "BMP: Bridging the Gap between B-Spline and Movement Primitives", in *CoRL 2024 Workshop on Learning Effective Abstractions for Planning*.

[14] Hongyi Zhou, Fabian Otto, Onur Celik, **Ge Li**, Rudolf Lioutikov, Gerhard Neumann, "MP3: Movement Primitive-Based (Re-) Planning Policies", in *CoRL 2023 Workshop on Learning Effective Abstractions for Planning*.

[15] Atalay Donat, Xiaogang Jia, Xi Huang, Aleksandar Taranovic, Denis Blessing, **Ge Li**, Hongyi Zhou, Hanyi Zhang, Rudolf Lioutikov, Gerhard Neumann. "Towards Fusing Point Cloud and Visual Representations for Imitation Learning". *arxiv,* 2025.

Supervised Student Theses

This section contains a complete list of the student theses that I have supervised during my PhD studies.

[1] Robin Wahl, "Task-Replanning in Transformer-based Episodic RL using Movement Primitives.", KIT, Master's Thesis, 2025.

[2] Serge Thilges, "Using Natural Policy Gradient with Trust Region for Fast and Stable State-dependent Full Covariance Gaussian Policy Training.", KIT, Master's Thesis, 2024.

[3] Dong Tian, "Deep Off-policy Reinforcement Learning with Transformer-based Trajectory-wise Q-function Prediction.", KIT, Master's Thesis, 2024.

[4] Max Nagy, "Solving Real-World Contact-rich Robot Manipulation Tasks with Deep Reinforcement Learning.", KIT, Master's Thesis, 2023.

[5] Weiran Liao, "Enhanced Probabilistic Dynamic Movement Primitives in Arbitrarily High-order Dynamic Systems.", KIT, Master's Thesis, 2023.

[6] Zeqi Jin, "Information Aggregation in Partial Observable MDPs for Deep Reinforcement Learning via Transformer and Recurrent Neural Networks.", KIT, Master's Thesis, 2023.

[7] Jan Küblbeck, "Benchmarking on Realistic Simulated Environments.", KIT, Bachelor's Thesis, 2022.

[8] Hongyi Zhou, "Robot Online Motion (Re)Planning with Episode-based Reinforcement Learning.", KIT, Master's Thesis, 2022.

[9] Roman Freiberg, "Sequencing of Motion Primitives using Off-Policy Reinforcement Learning.", KIT, Master's Thesis, 2021.

[10] Xin Tian, "Trajectory Segmentation for Robot Manipulation Datasets using Recurrent Neural Networks.", KIT, Bachelor's Thesis, 2021.

[11] Dongzhuoran Zhou, "Contextual Policy Search for Simulated Robot Table Tennis.", KIT, Master's Thesis, 2020.