TOP-ERL: Transformer-based Off-Policy Episodic Reinforcement Learning

Ge Li*, Dong Tian, Hongyi Zhou, Xinkai Jiang, Rudolf Lioutikov, Gerhard Neumann
Karlsruhe Institute of Technology, Germany

Abstract—**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. Our code is available here.**

I. Introduction

This work proposes a novel off-policy Reinforcement Learning (RL) algorithm that leverages a Transformer architecture to predict the value of action sequences. These predicted returns are then used to update a policy that outputs smooth trajectories, instead of selecting a single action at each decision step. Predicting full action trajectories is a key feature of episodic RL (ERL) [22], and it differs fundamentally from step-based RL (SRL) methods such as SAC [16], where decisions are made at each time step. Recent works have demonstrated the promise of ERL-style action selection in RL [35, 24], and similar trends have been observed in Imitation Learning, where predicting action sequences rather than individual actions has led to notable performance gains [63, 44]. Furthermore, episodic decision-making mirrors human behavior—people typically execute a coherent sequence of actions to complete a task (e.g., swinging an arm to hit a tennis ball), rather than planning each step individually.

Episodic RL represents a distinct branch of RL that focuses on optimizing cumulative returns over entire episodes, rather than intermediate steps [57, 19, 40]. Unlike SRL, ERL shifts the optimization space from per-step actions to parameterized trajectories, often leveraging methods such as Movement Primitives (Movement Primitives (MPs)) [49, 38] to generate smooth action sequences. This strategy offers several advantages: it enables a broader exploration horizon [22], captures temporal and joint correlations among degrees of freedom (DoF) [24], and facilitates smooth transitions between replanning phases [36]. With the integration of deep learning, ERL has recently demonstrated potential in tasks involving skill acquisition [9] and safety-aware learning in robotics [21].

However, a major limitation of current ERL approaches is their low data efficiency. Most methods remain confined to on-policy training, which restricts their ability to reuse past experiences. In contrast, off-policy methods—such as SAC—achieve higher sample efficiency by learning a critic (action-value function) that guides the policy via temporal difference (TD) learning [53]. Yet, TD learning assumes per-step action selection, making it incompatible with the sequence-based decision paradigm of ERL. This mismatch prevents ERL from fully benefiting from off-policy learning.

To overcome this limitation, we propose an approach that uses a Transformer to predict N-step returns [54] for sequences of actions. This formulation enables value learning and policy improvement in an off-policy setting, while remaining aligned with the trajectory-level structure of ERL.

Transformers in RL. Transformers [56] have become a foundational architecture for sequence modeling and have shown strong capabilities in temporal pattern recognition and long-term credit assignment. In RL, they have been successfully applied to offline RL [10, 59, 58], offline-to-online fine-tuning [64, 28, 62], partially observable environments [39, 32, 27], and model-based RL [26]. However, their integration into model-free, online RL—particularly for evaluating and predicting values of action sequences—remains underexplored [61]. This is surprising, as similar ideas like action chunking [6] have already shown effectiveness in imitation learning.

We propose **Transformer-based Off-Policy Episodic RL (TOP-ERL)**, which employs a Transformer as a critic to predict the return of action sequences. Given a full trajectory from an ERL policy, we segment it into smaller sequences and use the Transformer to predict their values. These value predictions are trained using N-step TD targets, allowing us to apply off-policy updates to the trajectory-generating policy. The policy then selects sequences of actions based on the Transformer's value estimates, analogous to SAC but operating in the trajectory space.

Our contributions are: (a) A novel off-policy episodic RL method that integrates a Transformer-based critic for action

*An RSS workshop version of the original paper, which was accepted at ICLR 2025. Email to <geli.bruce.ai@gmail.com>

sequences within a model-free, online framework; (b) The use of N-step return prediction as a training target for the Transformer critic; (c) Comprehensive evaluation on simulated robotic manipulation tasks, showing improved performance and sample efficiency over baseline SRL and ERL methods; (d) In-depth analysis of critic design choices, update rules, and the effect of trajectory segment length on performance.

II. Related Works

Episodic Reinforcement Learning (ERL). Research on ERL dates back to the 1990s, with early methods relying on black-box optimization to update policy parameters, typically small MLPs [57, 19, 14]. Due to their high data requirements and the limited computational resources of the time, these methods were restricted to low-dimensional tasks such as Pendulum and Cart Pole. Later works demonstrated that, given sufficient compute, ERL methods could match the performance of step-based RL approaches on more complex locomotion tasks (e.g., Ant, Humanoid), albeit with higher sample demands [48, 30].

Another research direction in ERL focuses on more compact policy representations. Peters and Schaal [40] introduced the use of movement primitives (MPs) as parameterized policies, reducing the optimization space from high-dimensional neural network parameters to MP weights (typically 20–50 dimensions). This significantly improves sample efficiency and provides additional benefits such as smooth trajectories and consistent exploration [24]. MP-based ERL methods have been applied to challenging manipulation tasks like robot baseball [40] and juggling [41]. To further improve data efficiency, Abdolmaleki et al. [1] proposed a model-based extension to support more sample-efficient black-box search. However, these methods struggle with contextual variation (e.g., changing goals). To address this, Abdolmaleki et al. [2] and Celik et al. [8] introduced context-conditioned policies, and Otto et al. [35] further improved this line by integrating neural network policies with trust-region-based updates.

Despite these advancements, most ERL methods treat the trajectory as a black box—an approach that enables robustness to sparse or non-Markovian rewards, but limits sample efficiency in dense-reward settings. To mitigate this, *Temporally-Correlated ERL* (TCE) [24] recently proposed a more efficient update scheme that incorporates sub-segment information for policy updates while preserving episodic exploration. However, TCE still relies on on-policy policy gradient updates, which are inherently sample-inefficient. To the best of our knowledge, TOP-ERL is the first ERL algorithm that supports off-policy training and can effectively handle contextual tasks.

Transformers in Model-Free RL. Motivated by the success of Transformers in sequential modeling, several studies have explored their integration into RL for tasks requiring long-horizon reasoning. However, directly applying standard Transformers to online RL often results in unstable training and poor performance, sometimes even comparable to a random policy [39]. To overcome this, *Gated Transformer-XL* (GTrXL) [39] introduced GRU-style gating between attention layers, stabilizing deep Transformer networks (up to 12 layers) in online RL.

Another line of work focuses on Transformers in offline RL, where learning is performed from a fixed dataset generated by arbitrary behavior policies. The *Decision Transformer* [11] pioneered this area by framing offline RL as a sequence modeling problem. This was extended with methods such as dynamic history truncation [58], Transformer-based Q-learning [59], and more efficient state-space model alternatives [33]. Online Decision Transformers [64] further adapted these models for online fine-tuning.

In contrast to the above approaches, which are predominantly designed for offline RL or fine-tuning, TOP-ERL is designed specifically for model-free, online RL and does not rely on pretraining or long-horizon memory handling. Instead, it leverages a Transformer critic to improve the estimation of action sequence values via multi-step TD learning within the ERL paradigm.

III. Preliminaries

A. Off-Policy Reinforcement Learning

Markov decision process (MDP). In reinforcement learning, an agent seeks a policy that maximizes cumulative reward in an environment modeled as an MDP. Formally, an MDP is the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$, where $\mathcal{S}$ and $\mathcal{A}$ are continuous state and action spaces, $P(s' \mid s, a)$ is the transition kernel, $r(s, a)$ the reward function, and $\gamma \in [0, 1]$ the discount factor. The objective is to find a policy $\pi(a \mid s)$ that maximizes the expected *return*, $G_t(s_t, a_t) = \sum_{i=0}^{\infty} \gamma^i r_{t+i}$.

Off-policy RL. Off-policy algorithms learn $\pi(a \mid s)$ from data generated by a separate behaviour policy $\pi_b(a \mid s)$, enabling efficient experience reuse. A *critic* estimates $Q^\pi(s, a)$ and is updated via the temporal-difference (TD) error

$$Q^\pi(s, a) = \mathbb{E}_\pi \left[G_t \mid s_t = s, a_t = a \right], \tag{1}$$

$$\delta_t = r_t + \gamma Q^\pi(s_{t+1}, a_{t+1}) - Q^\pi(s_t, a_t), \tag{2}$$

where δ_t measures the deviation between current and target Q-values. Although single-step TD updates are simple, they can be biased and propagate information slowly in delayed-reward settings. Employing N-step returns [53] often yields a better bias–variance trade-off.

N-step return. The N-step return generalizes the single-step TD target by summing rewards over N future steps before bootstrapping. Because it uses more actual rewards, it usually reduces bias compared with the 1-step target, though at the cost of higher variance. In off-policy learning, the future action sequence used in that sum often differs from the current policy $\pi(a \mid s)$, so importance sampling is applied [54]:

$$\begin{aligned} G_t^{(N)}(s_t, a_t) = &\sum_{i=0}^{N-1} \Big(\prod_{j=0}^{i} \rho_{t+j} \Big) \gamma^i \, r_{t+i} \\ &+ \Big(\prod_{j=0}^{N-1} \rho_{t+j} \Big) \gamma^N Q^\pi(s_{t+N}, a_{t+N}), \end{aligned} \tag{3}$$

where $\rho_t = \pi(a_t \mid s_t) / \pi_b(a_t \mid s_t)$ is the importance-sampling ratio that corrects for the behaviour policy π_b.

Practical challenges. For long horizons, the product of ratios in Eq.3 can explode or vanish, inflating variance and destabilizing training. TOP-ERL sidesteps this by computing $G_t^{(N)}(s_t, a_t, \ldots, a_{t+N})$ on fixed action sequences taken directly from the replay buffer, rather than from the current policy. As a result, importance weights are unnecessary. Details appear in Sec. IV-C.

B. *Episodic Reinforcement Learning (ERL)*

Episodic RL [57, 22] optimizes *entire* action sequences to maximize cumulative return, without evaluating each intermediate state transition. A parameterized trajectory generator—often a movement primitive (MP) [49, 38]—produces a parameter vector $\boldsymbol{w}$, which is mapped to a full action sequence

$$\boldsymbol{a}(\boldsymbol{w}) = \left[\boldsymbol{a}_t\right]_{t=0}^{T},$$

where $\boldsymbol{a}_t \in \mathbb{R}^D$ and D is the dimensionality of the action space (e.g., the robot's degrees of freedom). The agent can execute this sequence directly as motor commands or track it with a low-level controller.

Even though ERL reasons over multi-step trajectories, it still satisfies the *Markov property*: state transitions depend only on the current state and the executed action [54]. Thus ERL remains within the MDP framework. Conceptually, it relates to action-repeat strategies [7] and temporally correlated exploration [43, 12], which likewise embed temporal structure into action selection.

Movement Primitives (MPs) are compact, parameterised trajectory generators that underpin many ERL methods. Below we summarise the two variants used in this work; further details appear in App. D.

Schaal [49] introduced Dynamic Movement Primitives (DMPs), which augments a spring–damper system with a learnable forcing term to produce smooth trajectories from an initial condition[1]:

$$\tau^2 \ddot{y} = \alpha\big(\beta(g - y) - \tau \dot{y}\big) + f(x), \tag{4}$$

$$f(x) = x \frac{\sum \varphi_i(x) w_i}{\sum \varphi_i(x)} = x \boldsymbol{\varphi}_x^{\intercal} \boldsymbol{w}, \tag{5}$$

where $y(t)$, $\dot{y}(t)$, and $\ddot{y}(t)$ denote position, velocity, and acceleration. Parameters α and β control damping; g is the goal, and τ scales execution speed. The basis functions $\varphi_i(x)$ and weights w_i shape the forcing term. A trajectory $\left[y_t\right]_{t=0}^{T}$ is obtained by numerically integrating the system.

Building on these ideas, Li et al. [23] proposed Probabilistic Dynamic Movement Primitives (ProDMPs), which replaces numerical integration with a closed-form solution of Eq. 4. Using linear basis functions, ProDMP maps the parameter vector $\boldsymbol{w}$ directly to the trajectory:

$$y(t) = \boldsymbol{\Phi}(t)^{\intercal} \boldsymbol{w} + c_1 y_1(t) + c_2 y_2(t). \tag{6}$$

[1]An *initial condition* is the value of a function (or its derivatives) prescribed at a starting point, which need not be $t = 0$.

The terms $c_1 y_1(t) + c_2 y_2(t)$ guarantee exact satisfaction of the initial condition $(y_b, \dot{y}_b)$ at $t = t_b$, while $\boldsymbol{\Phi}(t)$ is the integral form of the basis $\boldsymbol{\varphi}$. Because ProDMP has a closed form, it supports faster rollout, straightforward probabilistic interpretation, and exact enforcement of initial conditions.

TOP-ERL exploits these advantages: ProDMP's fast, accurate rollouts let us compute low-bias target values for the Transformer critic, improving overall policy learning.

ERL learning objectives. A key difference between episodic RL (ERL) and step-based RL (SRL) is the action domain: ERL searches in the parameterised trajectory space $\mathcal{W}$, learning $\pi(\boldsymbol{w} \mid \boldsymbol{s})$, whereas SRL operates directly in the per-step action space $\mathcal{A}$. Each vector $\boldsymbol{w} \in \mathcal{W}$ specifies an entire trajectory and is therefore treated as a single data point. This naturally leads ERL to adopt black-box optimisation for trajectory search [48]. A common objective, used in BBRL [35], is

$$J = \mathbb{E}_{\pi_{\text{old}}(\boldsymbol{w}|\boldsymbol{s})}\left[\frac{\pi_{\text{new}}(\boldsymbol{w}|\boldsymbol{s})}{\pi_{\text{old}}(\boldsymbol{w}|\boldsymbol{s})} G^{\pi_{\text{old}}}(\boldsymbol{s}, \boldsymbol{w})\right], \tag{7}$$

where $\pi_{\boldsymbol{\theta}}$ is the current policy network, "new" denotes the policy being optimised, and "old" the policy that collected the data. The context $\boldsymbol{s} \in \mathcal{S}$ defines the task, and $G^{\pi_{\text{old}}}(\boldsymbol{s}, \boldsymbol{w}) = \sum_{t=0}^{T} \gamma^t r_t$ is the return obtained by executing $\boldsymbol{w}$ under the old policy.

Parameterised generators such as MPs promote smooth trajectories, consistent exploration, and resilience to local optima [35]. To improve data efficiency, TCE [24] decomposes each trajectory into K segments of length L, updating with segment-wise returns:

$$J = \mathbb{E}_{\pi_{\text{old}}(\boldsymbol{w}|\boldsymbol{s})}\left[\frac{1}{K} \sum_{k=1}^{K} \frac{p^{\pi_{\text{new}}}\big([\boldsymbol{a}_t^k]_{t=0:L} \mid \boldsymbol{s}\big)}{p^{\pi_{\text{old}}}\big([\boldsymbol{a}_t^k]_{t=0:L} \mid \boldsymbol{s}\big)} G^{\pi_{\text{old}}}\big(\boldsymbol{s}_0^k, [\boldsymbol{a}_t^k]_{t=0:L}\big)\right], \tag{8}$$

where K (set to 25 in the original work) is the number of segments and p^{π} the likelihood of reproducing a segment under policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{w} \mid \boldsymbol{s})$. Despite the importance-sampling ratios, both Eq. 7 and Eq. 8 remain on-policy objectives.

TOP-ERL adopts the same segmentation idea but performs *off-policy* updates, using replayed segments for critic and policy learning. This yields higher sample efficiency while retaining the benefits of trajectory-level exploration.

IV. 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 IV-A introduces the Gaussian policy modeling and action trajectory generation, followed by the design of the transformer critic in Section IV-B. The learning objectives for the critic and policy are detailed in Section IV-C and Section IV-E, respectively, with additional technical details. The main contributions of our model are described from Section IV-B to Section IV-E, while the remaining sections cover techniques adopted from the literature.

A. Trajectory Generation and Environment Rollout

TOP-ERL adopts a policy structure similar to previous ERL approaches, such as BBRL [35]. As shown in Fig. 1a, 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 the movement primitive (MPs). In TOP-ERL, we employ ProDMPs [23] to help correct the target computation via enforcing the initial condition of the MP, as discussed later in Section IV-D. 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.

B. Transformers as value predictor for action sequences

The architecture of our Transformer critic is depicted in Fig. 1b. 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. 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.

C. 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]

$$\mathcal{L}(\phi) = \frac{1}{L}\sum_{N=1}^{L-1}\left[\underbrace{Q_\phi(\boldsymbol{s}_0^k, \boldsymbol{a}_0^k, ..., \boldsymbol{a}_{N-1}^k)}_{\text{Predicted value of N actions}} - \underbrace{G^{(N)}(\boldsymbol{s}_0^k, \boldsymbol{a}_0^k, ..., \boldsymbol{a}_{N-1}^k)}_{\text{Target using } \textbf{N-step return}}\right]^2 + \left[\underbrace{V_\phi(\boldsymbol{s}_0^k)}_{\text{Predicted state value}} - \mathbb{E}_{\tilde{\boldsymbol{w}}\sim\pi_\theta(\cdot|s)}\underbrace{[Q_{\phi_{\text{tar}}}(\boldsymbol{s}_0^k, \tilde{\boldsymbol{a}}_0^k, ..., \tilde{\boldsymbol{a}}_{L-1}^k)]}_{\text{Target of new actions using } \tilde{\boldsymbol{w}}}\right]^2, \tag{9}$$

[2]For simplicity, we omit the expectation over buffer $\mathcal{B}$ and average over segment number K in Eq.(9).

where the **N-step return** is formulated as:

$$G^{(N)}(\boldsymbol{s}_0^k, \boldsymbol{a}_0^k, ..., \boldsymbol{a}_{N-1}^k) = \underbrace{\sum_{i=0}^{N-1}\gamma^i r_i}_{\text{N-step rewards}} + \underbrace{\gamma^N V_{\phi_{\text{tar}}}(\boldsymbol{s}_N).}_{\text{Future return after N-step}} \tag{10}$$

Here, $N \in [1, L-1]$ represents the number of actions in a subsequence 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.(9) 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.(10) represents the future return after N steps. Both $Q_{\phi_{\text{tar}}}$ and $V_{\phi_{\text{tar}}}$ are predicted by a target critic [31], 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.(10). 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 [13], to be stable and effective. We attribute this to the usage of the N-step returns, which help reduce value estimation bias.

Unlike Eq.(3), our N-step return targets $G^{(N)}(\boldsymbol{s}_0^k, \boldsymbol{a}_0^k, \ldots, \boldsymbol{a}_{N-1}^k)$ in Eq.(10) 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.(10) 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. III-A.

D. Enforce initial condition for newly predicted actions

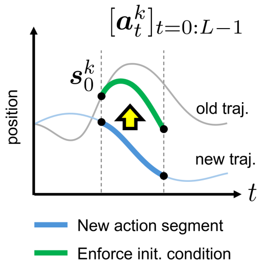
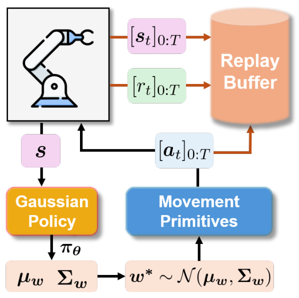
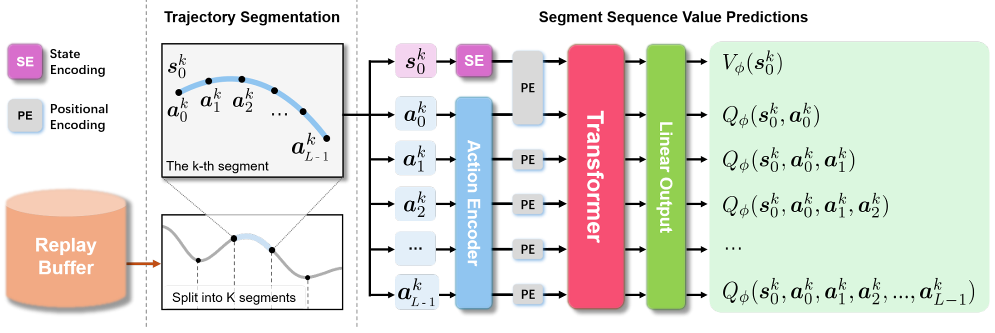


Fig. 2: Enforce action initial condition

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.(9), a new parameter vector is sampled from the current policy $\tilde{\boldsymbol{w}} \sim \pi_\theta(\ \cdot\ |\boldsymbol{s})$, generating a new action trajectory $[\tilde{\boldsymbol{a}}_t]_{t=0:T}$, with $[\tilde{\boldsymbol{a}}_t^k(\tilde{\boldsymbol{w}})]_{t=0:L-1}$ as a subsequence. 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.(9). 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. 2. 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



(a)


SE State Encoding
PE Positional Encoding
Replay Buffer
Trajectory Segmentation
s_0^k a_0^k a_1^k a_2^k ... a_{L-1}^k
The k-th segment
Split into K segments
Segment Sequence Value Predictions
Action Encoder
Transformer
Linear Output
$V_\phi(s_0^k)$
$Q_\phi(s_0^k, a_0^k)$
$Q_\phi(s_0^k, a_0^k, a_1^k)$
$Q_\phi(s_0^k, a_0^k, a_1^k, a_2^k)$
...
$Q_\phi(s_0^k, a_0^k, a_1^k, a_2^k, ..., a_{L-1}^k)$

(b)

Fig. 1: (a) Trajectory generation and environment rollout. (b) Transformer critic architecture, as described in Sec. IV-B.

state and action sequences. Further mathematical details and illustration are provided in Appendix G and H.

E. 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 [16]. The learning objective is to maximize the expected value of the averaged action sequence over varying lengths, defined as:

$$J(\theta) = \mathbb{E}_{s\sim B}\mathbb{E}_{\tilde{w}\sim\pi_\theta(\cdot|s)}\left[\frac{1}{KL}\sum_{k=1}^{K}\sum_{N=0}^{L-1} Q_\phi(s_0^k, \left[\tilde{a}_t^k\right]_{t=0:N})\right], \tag{11}$$

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

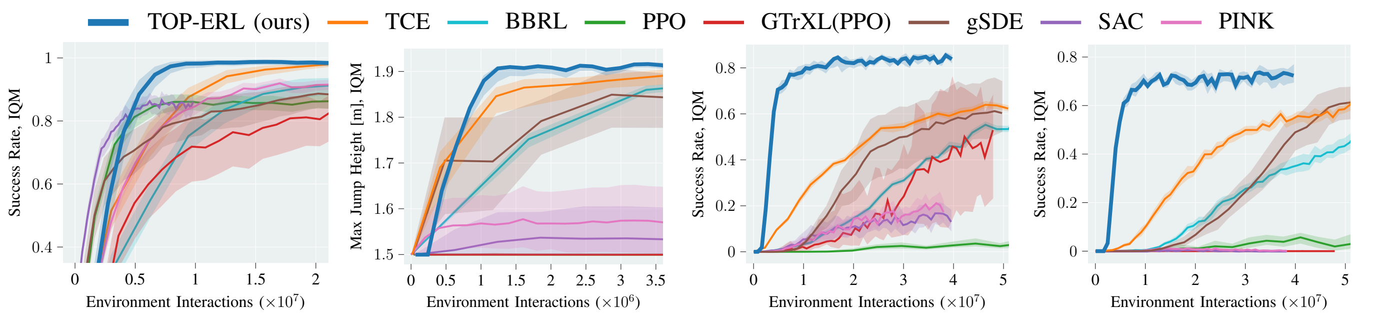
V. Experiments

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** [51] (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 [25], 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 II, and details regarding the implementation and hyperparameters are provided in the Appendix M.

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 [24]. Next, we evaluate TOP-ERL on the Meta-World MT50 [60] 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 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 [3] across 8 random seeds.

A. 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. [24]: 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 H. The results of these experiments, shown in Fig. 3, demonstrate that TOP-ERL achieved the highest final performance across all three tasks. Notably, in the dense-reward Box Pushing task, TOP-ERL



(a) Metaworld Agg. of 50 tasks (b) Hopper Jump Max Height (c) Box Pushing Dense Reward (d) Box Pushing Sparse Reward

Fig. 3: 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.

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.

B. 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[60], 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 [35] and [24], 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. 3a 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.

VI. Conclusion

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.

References

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

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

[3] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in Neural Information Processing Systems*, 2021.

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

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

[6] 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)*, pages 4788–4795. IEEE, 2024.

[7] 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.

[8] 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*, pages 1423–1433. PMLR, 2022.

[9] Onur Celik, Aleksandar Taranovic, and Gerhard Neumann. Acquiring diverse skills using curriculum reinforcement learning with mixture of experts. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=9ZkUFSwlUH.

[10] 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*, pages 3909–3928. PMLR, 2023.

[11] 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. *Advances in neural information processing systems*, 34:15084–15097, 2021.

[12] 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.

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

[14] Faustino Gomez, Jürgen Schmidhuber, Risto Miikkulainen, and Melanie Mitchell. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9(5), 2008.

[15] 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)*, pages 502–508. IEEE, 2016.

[16] 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*, pages 1861–1870. PMLR, 2018.

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

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

[19] Christian Igel. Neuroevolution for reinforcement learning using evolution strategies. In *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.*, volume 4, pages 2588–2595. IEEE, 2003.

[20] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[21] 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.

[22] Jens Kober and Jan Peters. Policy search for motor primitives in robotics. *Advances in neural information processing systems*, 21, 2008.

[23] Ge Li, Zeqi Jin, Michael Volpp, Fabian Otto, Rudolf Lioutikov, and Gerhard Neumann. Prodmp: A unified perspective on dynamic and probabilistic movement primitives. *IEEE Robotics and Automation Letters*, 8(4): 2325–2332, 2023.

[24] 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.

[25] 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*, pages 3053–3062. PMLR, 2018.

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

[27] 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.

[28] 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)*, pages 10793–10799. IEEE, 2024.

[29] 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*, pages 527–534. IEEE, 2014.

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

[31] 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. *nature*, 518(7540):529–533, 2015.

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

[33] Toshihiro Ota. Decision mamba: Reinforcement learning via sequence modeling with selective state spaces. *arXiv preprint arXiv:2403.19925*, 2024.

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

[35] 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*, pages 1244–1265. PMLR, 2022.

[36] Fabian Otto, Hongyi Zhou, Onur Celik, Ge Li, Rudolf Lioutikov, and Gerhard Neumann. Mp3: Movement primitive-based (re-) planning policy. *arXiv preprint arXiv:2306.12729*, 2023.

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

[38] Alexandros Paraschos, Christian Daniel, Jan Peters, and Gerhard Neumann. Probabilistic movement primitives. *Advances in neural information processing systems*, 26, 2013.

[39] 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*, pages 7487–7498. PMLR, 2020.

[40] Jan Peters and Stefan Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21 (4):682–697, 2008.

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

[42] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

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

[44] 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.

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

[46] Thomas Rückstieß, Martin Felder, and Jürgen Schmidhuber. State-dependent exploration for policy gradient methods. In Walter Daelemans, Bart Goethals, and Katharina Morik, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 234–249, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. ISBN 978-3-540-87481-2.

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

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

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

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

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

[52] 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)*, pages 2666–2671. IEEE, 2019.

[53] Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

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

[55] Jens Timmer and Michel Koenig. On generating power law noise. *Astronomy and Astrophysics*, 300:707, 1995.

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

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

[58] Yueh-Hua Wu, Xiaolong Wang, and Masashi Hamaya. Elastic decision transformer. *Advances in Neural Information Processing Systems*, 36, 2024.

[59] 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*, pages 38989–39007. PMLR, 2023.

[60] 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*, pages 1094–1100. PMLR, 2020.

[61] Weilin Yuan, Jiaxing Chen, Shaofei Chen, Dawei Feng, Zhenzhen Hu, Peng Li, and Weiwei Zhao. Transformer in reinforcement learning for decision-making: a survey. *Frontiers of Information Technology & Electronic Engineering*, 25(6):763–790, 2024.

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

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

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

[65] 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)*, pages 4301–4308. IEEE, 2019.

Algorithm 1 TOP-ERL

```
 1: Initialize critic $\phi$; target critic $\phi_{\text{tar}} \leftarrow \phi$
 2: Initialize policy $\theta$ and replay buffer $\mathcal{B}$
 3: repeat
 4:     Reset environment and get initial task state $\boldsymbol{s}$
 5:     Predict the policy mean $\boldsymbol{\mu_w}$ and covariance $\boldsymbol{\Sigma_w}$
 6:     Sample $\boldsymbol{w}*$ and generate action trajectory $[\boldsymbol{a}]_{0:T}$
 7:     Execute the action trajectory till task ends.
 8:     Store the visited states $[\boldsymbol{s}]_{0:T}$, rewards $[r]_{0:T}$, and
          the action $[\boldsymbol{a}]_{0:T}$ trajectories in replay buffer $\mathcal{B}$
 9:     for each update step do
10:         From $\mathcal{B}$, sample a batch of $\boldsymbol{s}, \boldsymbol{a}, r$ trajectories.
11:         Split them into $K$ segments, each $L$ time steps.
12:         Compute N-step return targets as in Eq.(10)
13:         Update transformer critic, using Eq.(9)
14:         Update policy, using Eq.(11)
15:     end for
16:     Update target critic $\phi_{\text{tar}} \leftarrow (1-\rho)\ \phi_{\text{tar}} + \rho\ \phi$
17: until converged
```

Appendix

A. Algorithm Box

We summarize the key learning steps in Algorithm1.

B. 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 [16]. Given a task initial state $\boldsymbol{s}$, the current policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{w}|\boldsymbol{s}) \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu_w}, \boldsymbol{\Sigma_w})$ predicts the Gaussian parameters of the MP's and samples $\tilde{\boldsymbol{w}}$ as follows:

$$\tilde{\boldsymbol{w}} = \boldsymbol{\mu_w} + \boldsymbol{L_w}\boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}). \tag{12}$$

Here, $\boldsymbol{L_w}$ is the Cholesky decomposition of the covariance matrix $\boldsymbol{\Sigma_w}$, where $\boldsymbol{L_w}\boldsymbol{L_w}^T = \boldsymbol{\Sigma_w}$. 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. (12) represents the full covariance extension of the reparameterization trick typically used in RL, known as $\tilde{a} = \mu_a + \sigma_a \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)$.

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. 22 in Appendix G:

$$\tilde{\boldsymbol{a}}(t) = \boldsymbol{\Phi}(t)^{\intercal}\tilde{\boldsymbol{w}} + c_1 y_1(t) + c_2 y_2(t) \tag{4}$$

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.

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:

$$J(\theta) = \mathbb{E}_{\boldsymbol{s}\sim B}\mathbb{E}_{\tilde{\boldsymbol{w}}\sim\pi_\theta(\cdot|\boldsymbol{s})}\left[\frac{1}{KL}\sum_{k=1}^{K}\sum_{N=0}^{L-1} Q_\phi(\boldsymbol{s}_0^k, [\tilde{\boldsymbol{a}}_t^k]_{t=0:N})\right]. \tag{9}$$

Since Eq.(4), (9), (22) and (12) 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. Utilizing TRPL for Stable Full-Covariance Gaussian Policy Training

In Episodic Reinforcement Learning (ERL), the parameter space $\mathcal{W}$ generally has a higher dimensionality than the action space $\mathcal{A}$, creating distinct challenges for achieving stable policy updates. Trust region methods [50, 51] are widely regarded as reliable techniques for ensuring convergence and stability in policy gradient algorithms.

Although methods like PPO approximate trust regions using surrogate objectives, they lack the ability to enforce trust regions precisely. To address this limitation, Otto et al. [34] proposed trust region projection layer (TRPL), a mathematically rigorous and scalable approach for exact trust region enforcement in deep RL algorithms. Leveraging differentiable convex optimization layers [4], trust region projection layer (TRPL) enforces trust regions at the per-state level and has demonstrated robustness and stability in high-dimensional parameter spaces, as evidenced in methods like BBRL [35] and TCE [24].

TRPL operates on the standard outputs of a Gaussian policy—the mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$—and enforces trust regions through a state-specific projection operation. The adjusted Gaussian policy, represented by $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\Sigma}}$, serves as the foundation for subsequent computations. The dissimilarity measures for the mean and covariance, denoted as d_{mean} and d_{cov} (e.g., KL-divergence), are bounded by thresholds ϵ_μ and ϵ_Σ, respectively. The optimization problem for each state $\boldsymbol{s}$ is expressed as:

$$\begin{aligned}
&\arg\min_{\tilde{\boldsymbol{\mu}}_s} d_{\text{mean}}\left(\tilde{\boldsymbol{\mu}}_s, \boldsymbol{\mu}(\boldsymbol{s})\right), \quad \text{s.t.} \quad d_{\text{mean}}\left(\tilde{\boldsymbol{\mu}}_s, \boldsymbol{\mu}_{\text{old}}(\boldsymbol{s})\right) \le \epsilon_\mu, \\
&\arg\min_{\tilde{\boldsymbol{\Sigma}}_s} d_{\text{cov}}\left(\tilde{\boldsymbol{\Sigma}}_s, \boldsymbol{\Sigma}(\boldsymbol{s})\right), \quad \text{s.t.} \quad d_{\text{cov}}\left(\tilde{\boldsymbol{\Sigma}}_s, \boldsymbol{\Sigma}_{\text{old}}(\boldsymbol{s})\right) \le \epsilon_\Sigma.
\end{aligned} \tag{13}$$

If the unconstrained, newly predicted per-state Gaussian parameters $\boldsymbol{\mu}(\boldsymbol{s})$ and $\boldsymbol{\Sigma}(\boldsymbol{s})$ exceed the trust region bounds defined by ϵ_μ and ϵ_Σ, respectively, TRPL projects them back to the trust region boundary, ensuring stable update steps. In TOP-ERL, the old Gaussian parameters, $\boldsymbol{\mu}_{\text{old}}(\boldsymbol{s})$ and $\boldsymbol{\Sigma}_{\text{old}}(\boldsymbol{s})$, can be derived either from the behavior policy that interacted with the environment or from an exponentially moving averaged (EMA) policy, which serves as a delayed version of the current policy. This approach is analogous to the concept employed in the target critic network.

D. Target Options

TABLE I: Options for the future return used in Eq.(10)

Option	Math	Description
V-target	$V_\phi^{\text{tar}}(\boldsymbol{s}_N)$	State value after N steps
Q-target	$Q_\phi^{\text{tar}}(\boldsymbol{s}_N, \boldsymbol{a}_N, ...)$	Action value after N steps
Clipped	$\text{Min}(\cdot,\cdot)$	Minimum of 2 target critics
Ensemble	$\text{Avg.}(\cdot,\cdot)$	Mean of ≥ 2 target critics

In this section, we provide an overview of the movement primitive formulations used in this paper. We begin with the basics of DMPs and ProMPs, followed by a detailed explanation of ProDMPs. For clarity, we start with a single DoF system and then expand to multi-DoF systems.

E. Dynamic Movement Primitives

Schaal [49], Ijspeert et al. [20] describe a single movement as a trajectory $[y_t]_{t=0:T}$, which is governed by a second-order linear dynamical system with a non-linear forcing function f. The mathematical representation is given by

$$\tau^2\ddot{y} = \alpha(\beta(g-y) - \tau\dot{y}) + f(x), \tag{14}$$

$$f(x) = x\frac{\sum\varphi_i(x)w_i}{\sum\varphi_i(x)} = x\boldsymbol{\varphi}_x^\intercal\boldsymbol{w}, \tag{15}$$

where $y = y(t)$, $\dot{y} = \mathrm{d}y/\mathrm{d}t$, $\ddot{y} = \mathrm{d}^2y/\mathrm{d}t^2$ denote the position, velocity, and acceleration of the system at a specific time t, respectively. Constants α and β are spring-damper parameters, g signifies a goal attractor, 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 $x(t) = \exp(-\alpha_x/\tau; t)$. Here, $\varphi_i(x)$ represents the basis functions for the forcing term. The trajectory's shape as it approaches the goal is determined by the weight parameters $w_i \in \boldsymbol{w}$, for $i = 1, ..., N$. The trajectory $[y_t]_{t=0:T}$ is typically computed by numerically integrating the dynamical system from the start to the end point [37, 5]. However, this numerical process is computationally intensive. For example, to compute the trajectory segment in the end of an episode, DMP must integrate the system from the very beginning till the start of the segment.

F. Probabilistic Movement Primitives

Paraschos et al. [38] introduced a framework for modeling MPs using trajectory distributions, capturing both temporal

and inter-dimensional correlations. Unlike DMPs that use a forcing term, ProMPs directly model the intended trajectory. The probability of observing a 1-DoF trajectory $[y_t]_{t=0:T}$ given a specific weight vector distribution $p(\boldsymbol{w}) \sim \mathcal{N}(\boldsymbol{w}|\boldsymbol{\mu}_{\boldsymbol{w}}, \boldsymbol{\Sigma}_{\boldsymbol{w}})$ is represented as a linear basis function model:
Linear basis function:

$$[y_t]_{t=0:T} = \boldsymbol{\Phi}_{0:T}^{\intercal}\boldsymbol{w} + \epsilon_y, \tag{16}$$

Mapping distribution:

$$p([y_t]_{t=0:T};\ \boldsymbol{\mu}_{\boldsymbol{y}}, \boldsymbol{\Sigma}_{\boldsymbol{y}}) = \mathcal{N}(\boldsymbol{\Phi}_{0:T}^{\intercal}\boldsymbol{\mu}_{\boldsymbol{w}},\ \boldsymbol{\Phi}_{0:T}^{\intercal}\boldsymbol{\Sigma}_{\boldsymbol{w}}\boldsymbol{\Phi}_{0:T}\ + \sigma_{\boldsymbol{y}}^2\boldsymbol{I}). \tag{17}$$

Here, ϵ_y is zero-mean white noise with variance σ_y^2. The matrix $\boldsymbol{\Phi}_{0:T}$ houses the basis functions for each time step t. Similar to DMPs, these basis functions can be defined in terms of a phase variable instead of time. ProMPs allows for flexible manipulation of MP trajectories through probabilistic operators applied to $p(\boldsymbol{w})$, such as conditioning, combination, and blending [29, 15, 52, 45, 65]. However, ProMPs lack an intrinsic dynamic system, which means they cannot guarantee a smooth transition from the robot's initial state or between different generated trajectories.

G. Probabilistic Dynamic Movement Primitives

a) Solving the ODE underlying DMPs: Li et al. [23] noted that the governing equation of DMPs, as specified in Eq. (15), admits an analytical solution. This is because it is a second-order linear non-homogeneous ODE with constant coefficients. The original ODE and its homogeneous counterpart can be expressed in standard form as follows:

$$\text{Non-homo. ODE:}\quad \ddot{y} + \frac{\alpha}{\tau}\dot{y} + \frac{\alpha\beta}{\tau^2}y = \frac{f(x)}{\tau^2} + \frac{\alpha\beta}{\tau^2}g \equiv F(x,g), \tag{18}$$

$$\text{Homo. ODE:}\quad \ddot{y} + \frac{\alpha}{\tau}\dot{y} + \frac{\alpha\beta}{\tau^2}y = 0. \tag{19}$$

The solution to this ODE is essentially the position trajectory, and its time derivative yields the velocity trajectory. These are formulated as:

$$y = \begin{bmatrix} y_2\boldsymbol{p_2} - y_1\boldsymbol{p_1} & y_2q_2 - y_1q_1 \end{bmatrix}\begin{bmatrix}\boldsymbol{w}\\ g\end{bmatrix} + c_1y_1 + c_2y_2 \tag{20}$$

$$\dot{y} = \begin{bmatrix} \dot{y}_2\boldsymbol{p_2} - \dot{y}_1\boldsymbol{p_1} & \dot{y}_2q_2 - \dot{y}_1q_1 \end{bmatrix}\begin{bmatrix}\boldsymbol{w}\\ g\end{bmatrix} + c_1\dot{y}_1 + c_2\dot{y}_2. \tag{21}$$

Here, the learnable parameters $\boldsymbol{w}$ and g which control the shape of the trajectory, are separable from the remaining terms. Time-dependent functions $y_1, y_2, \boldsymbol{p}_1$, $\boldsymbol{p}_2$, q_1, q_2 in the remaining terms offer the basic support to generate the trajectory. The functions y_1, y_2 are the complementary solutions to the homogeneous ODE presented in equation 19, and $\dot{y}_1, \dot{y}_2$ their time derivatives respectively. These time-dependent functions take the form as:

$$\begin{aligned}
y_1(t) &= \exp\Big(-\frac{\alpha}{2\tau}t\Big),\\
y_2(t) &= t\exp\Big(-\frac{\alpha}{2\tau}t\Big),\\
\boldsymbol{p}_1(t) &= \frac{1}{\tau^2}\int_0^t t'\exp\Big(\frac{\alpha}{2\tau}t'\Big)x(t')\boldsymbol{\varphi}_x^{\intercal}\mathrm{d}t',\\
\boldsymbol{p}_2(t) &= \frac{1}{\tau^2}\int_0^t \exp\Big(\frac{\alpha}{2\tau}t'\Big)x(t')\boldsymbol{\varphi}_x^{\intercal}\mathrm{d}t',\\
q_1(t) &= \Big(\frac{\alpha}{2\tau}t - 1\Big)\exp\Big(\frac{\alpha}{2\tau}t\Big) + 1\\
q_2(t) &= \frac{\alpha}{2\tau}\Big[\exp\Big(\frac{\alpha}{2\tau}t\Big) - 1\Big].
\end{aligned}$$

It's worth noting that the $\boldsymbol{p}_1$ and $\boldsymbol{p}_2$ cannot be analytically derived due to the complex nature of the forcing basis terms $\boldsymbol{\varphi}_x$. As a result, they need to be computed numerically. Despite this, isolating the learnable parameters, namely $\boldsymbol{w}$ and g, allows for the reuse of the remaining terms across all generated trajectories. These residual terms can be more specifically identified as the position and velocity basis functions, denoted as $\boldsymbol{\Phi}(t)$ and $\dot{\boldsymbol{\Phi}}(t)$, respectively. When both $\boldsymbol{w}$ and g are included in a concatenated vector, represented as $\boldsymbol{w}_g$, the expressions for position and velocity trajectories can be formulated in a manner akin to that employed by ProMPs:

$$\textbf{Position:}\quad y(t) = \boldsymbol{\Phi}(t)^{\intercal}\boldsymbol{w}_g + c_1y_1(t) + c_2y_2(t), \tag{22}$$

$$\textbf{Velocity:}\quad \dot{y}(t) = \dot{\boldsymbol{\Phi}}(t)^{\intercal}\boldsymbol{w}_g + c_1\dot{y}_1(t) + c_2\dot{y}_2(t). \tag{23}$$

In the main paper, for simplicity and notation convenience, we use $\boldsymbol{w}$ instead of $\boldsymbol{w}_g$ to describe the parameters and goal of ProDMPs.

b) Intial Condition Enforcement: The coefficients c_1 and c_2 serve as solutions to the initial value problem delineated by the Eq.(22)(23). Li et al. propose utilizing the robot's initial state or the replanning state, characterized by the robot's position and velocity $(y_b, \dot{y}_b)$ to ensure a smooth commencement or transition from a previously generated trajectory. Denote the values of the complementary functions and their derivatives at the condition time t_b as $y_{1_b}, y_{2_b}, \dot{y}_{1_b}$ and $\dot{y}_{2_b}$. Similarly, denote the values of the position and velocity basis functions at this time as $\boldsymbol{\Phi}_b$ and $\dot{\boldsymbol{\Phi}}_b$ respectively. Using these notations, c_1 and c_2 can be calculated as follows:

$$\begin{bmatrix}c_1\\ c_2\end{bmatrix} = \begin{bmatrix} \frac{\dot{y}_{2_b}y_b - y_{2_b}\dot{y}_b}{y_{1_b}\dot{y}_{2_b} - y_{2_b}\dot{y}_{1_b}} + \frac{y_{2_b}\dot{\boldsymbol{\Phi}}_b^{\intercal} - \dot{y}_{2_b}\boldsymbol{\Phi}_b^{\intercal}}{y_{1_b}\dot{y}_{2_b} - y_{2_b}\dot{y}_{1_b}}\boldsymbol{w}_g \\ \frac{y_{1_b}\dot{y}_b - \dot{y}_{1_b}y_b}{y_{1_b}\dot{y}_{2_b} - y_{2_b}\dot{y}_{1_b}} + \frac{\dot{y}_{1_b}\boldsymbol{\Phi}_b^{\intercal} - y_{1_b}\dot{\boldsymbol{\Phi}}_b^{\intercal}}{y_{1_b}\dot{y}_{2_b} - y_{2_b}\dot{y}_{1_b}}\boldsymbol{w}_g \end{bmatrix}. \tag{24}$$

Despite the complex form used in the initial condition enforcement, the solutions conducted above only rely on solving several linear equations and can be easily implemented in a batch-manner and is therefore computationally efficient, normally ≤ 1 ms.

H. Enforce trajectory segments' initial condition in TOP-ERL

In Figure 4, we illustrate how TOP-ERL leverages this mechanism. The figure is based on the motion trajectory of

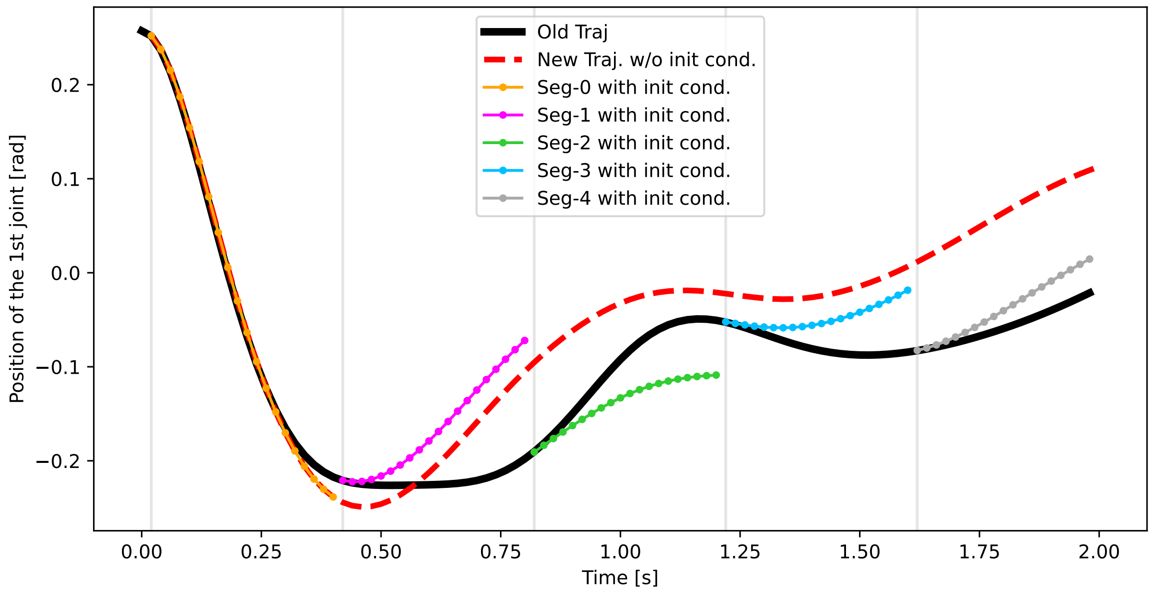


Fig. 4: TOP-ERL leverages the initial condition enforcement techniques of a dynamic system to ensure that the new action trajectory starts from the corresponding old state. **These action trajectories are taken from the first DoF of the robot in the box pushing task.**

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. 4, 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.

I. *Details of Methods Implementation*

a) PPO: Proximal Policy Optimization (PPO) [51] 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 [42].

b) SAC: Soft Actor-Critic (SAC) [16, 17] 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 [42].

c) GTrXL: Gated TransformerXL (GTrXL) [39] 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 [25]. We augmented the implementation with minibatch advantage normalization and state-independent log standard deviation as suggested in [18].

d) gSDE: Generalized State Dependent Exploration (gSDE) [43, 46, 47] 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

TABLE II: Baseline methods categorized by type (ERL or SRL) and update rules (On- or Off-policy).

Method	**Category**	**Description**
BBRL [35]	ERL, On	Black Box Optimization style ERL, policy search in parameter space
TCE [24]	ERL, On	Extend BBRL to use per-step info for efficient policy update
PPO [51]	SRL, On	Standard on-policy method with simplified Trust Region enforcement
gSDE [43]	SRL, On	Consecutive exploration noise for NN parameters of the policy
GTrXL[39]	SRL, On	Transformer-augmented SRL with multiple state as history
SAC [16]	SRL, Off	Standard off-policy method with entropy bonus for better exploration
PINK [12]	SRL, Off	Use temporal correlated pink noise for better exploration

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

$$\epsilon_t = \theta_\epsilon s, \text{ where } \theta_\epsilon \sim \mathcal{N}^d(0, \Sigma).$$

The exploration matrix θ_ϵ is composed of vectors of length $\mathrm{Dim}(a)$ 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. [43]. 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.

e) PINK: We utilize SAC to evaluate the effectiveness of pink noise for efficient exploration. Eberhard et al. [12] propose to replace the independent action noise ϵ_t of

$$a_t = \mu_t + \sigma_t \cdot \epsilon_t$$

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 [12].

We follow the reference implementation and sample chunks of Gaussian pink noise using the inverse Fast Fourier Transform method proposed by Timmer and Koenig [55]. 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.

f) BBRL: Black-Box Reinforcement Learning (BBRL) [35, 36] is a recent developed episodic reinforcement learning method. By utilizing ProMPs [38] 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 [23], aiming to isolate the effects attributable to different MP approaches.

g) TCE: Temporally-Correlated Episodic RL (TCE) [24] is an innovative ERL algorithm that leverages step-level information in episodic policy updates, shedding light on the 'black box' of current ERL methods while preserving smooth and consistent exploration within the parameter space. TCE integrates the strengths of both step-based and episodic RL, offering performance on par with recent ERL approaches, while matching the data efficiency of state-of-the-art (SoTA) step-based RL methods.

J. Metaworld

MetaWorld [60] is an open-source simulated benchmark specifically designed for meta-reinforcement learning and multi-task learning in robotic manipulation. It features 50 distinct manipulation tasks, each presenting unique challenges that require robots to learn a wide range of skills, such as grasping, pushing, and object placement. Unlike benchmarks that focus on narrow task distributions, MetaWorld provides a broader range of tasks, making it an ideal platform for developing algorithms that can generalize across different behaviors.

To ensure a fair comparison, we followed the evaluation protocol described in [35] and [24], where an episode is considered successful only if the success criterion is met at the end of the episode. This is equivalent to requiring the robot to complete the task and maintain its success state until the episode ends, which is a more rigorous measure than the original setting, where success at any time step is sufficient.

We reported each individual Metaworld task in Fig. 7 and Fig. 8. These tasks cover a wide range of types and complexities.

K. Hopper Jump

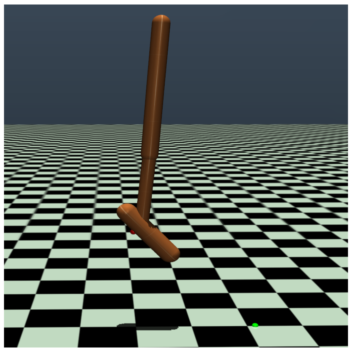


Fig. 5: Hopper Jump

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. 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.

L. Box Pushing

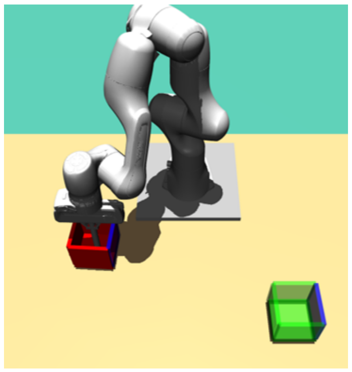


Fig. 6: Box Pushing

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 [35]. 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.

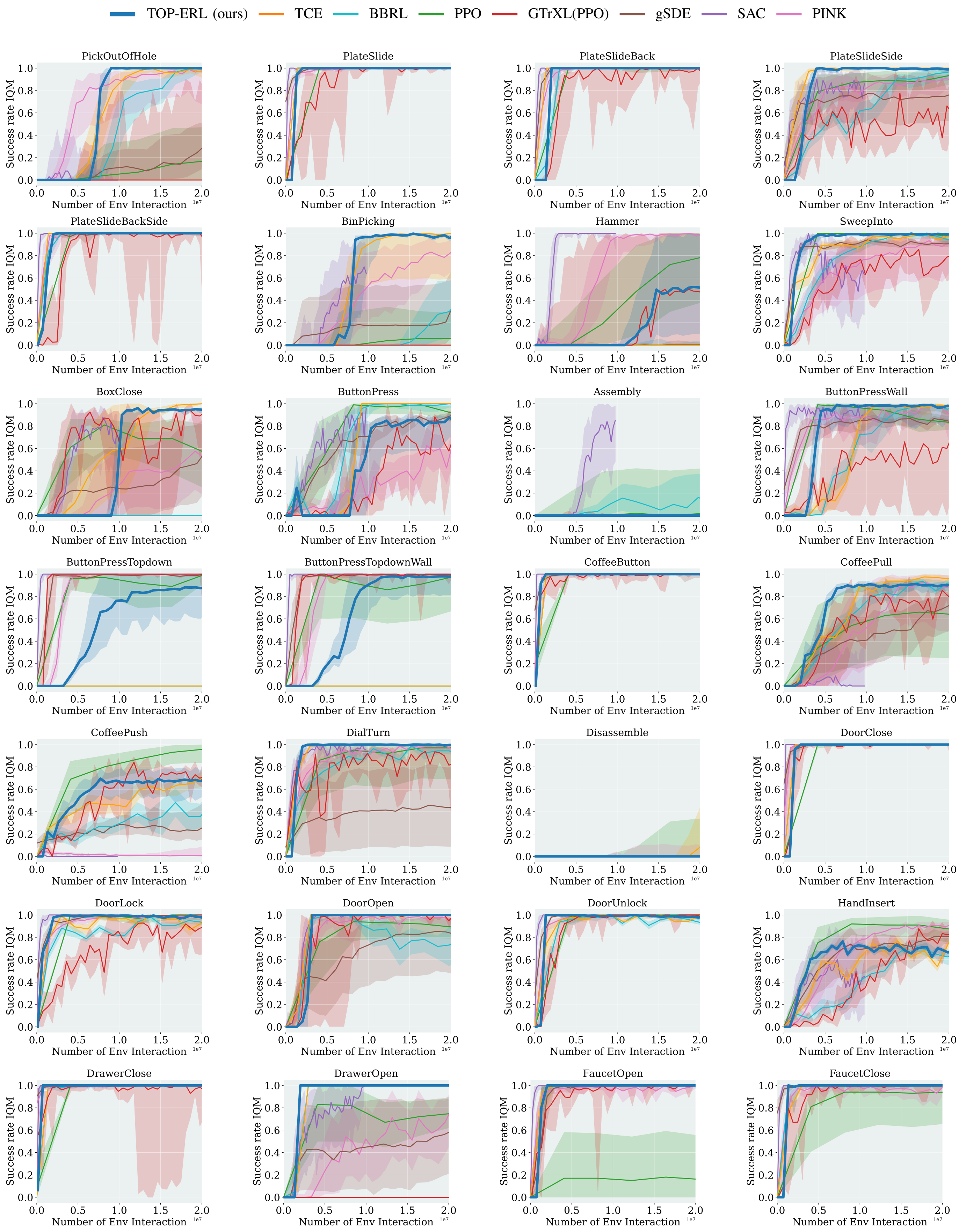


Fig. 7: Success Rate IQM of each individual Metaworld tasks.

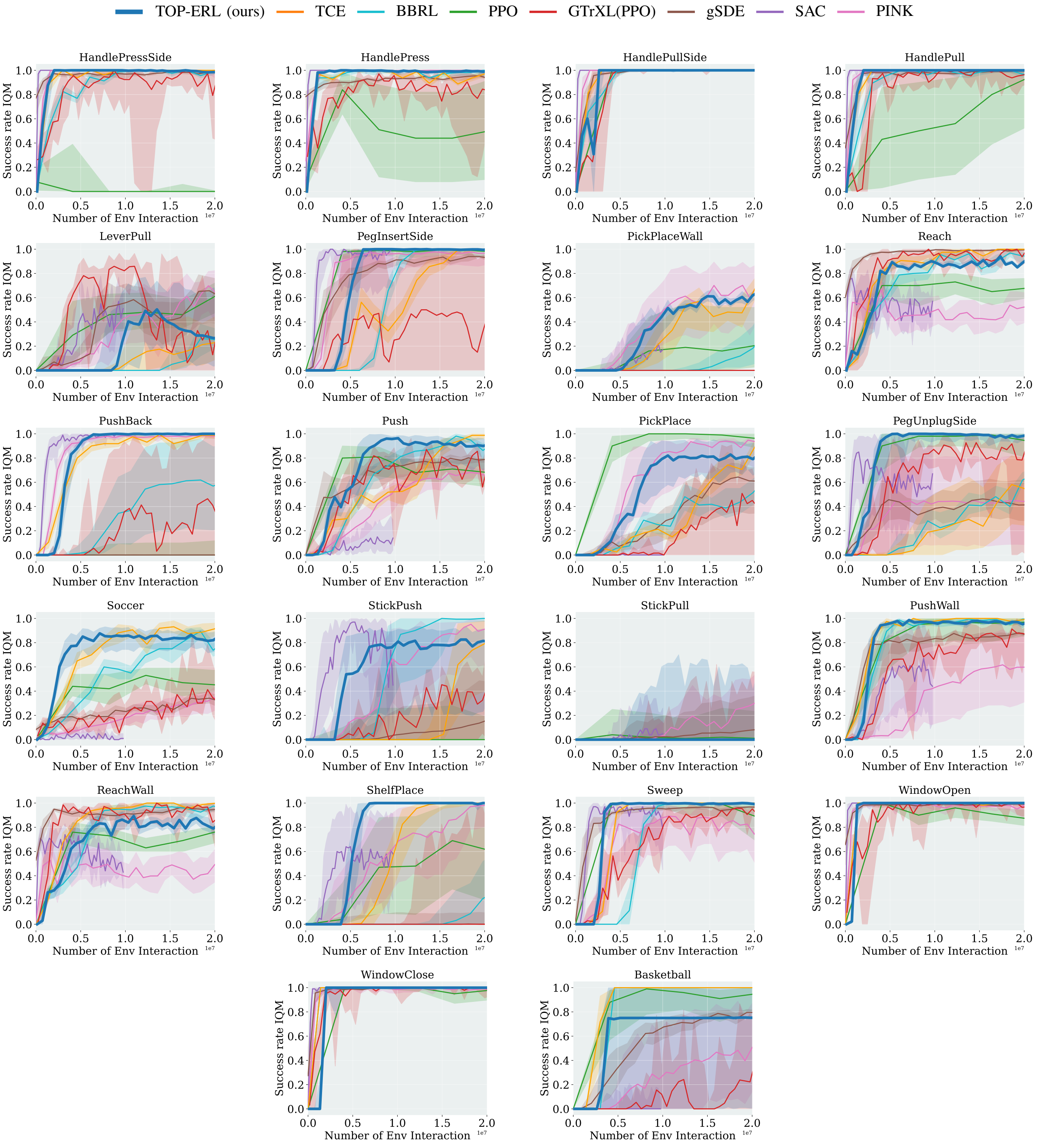


Fig. 8: Success Rate IQM of each individual Metaworld tasks.

M. Hyper Parameters

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) 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.

b) 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.

c) PPO:: Policy network size, critic network size, policy learning rate, critic learning rate, batch size, samples per iteration.

d) gSDE:: Same types of hyper-parameters listed in PPO, together with the state dependent exploration sampling frequency [43].

e) SAC:: Policy network size, critic network size, policy learning rate, critic learning rate, alpha learning rate, batch size, Update-To-Data (UTD) ratio.

f) PINK:: Same types of hyper-parameters listed in SAC.

g) 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

h) 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 hyper parameters 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 III: Hyperparameters 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
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
orthogonal initialization	Yes	No	xavier	fanin	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

TABLE IV: Hyperparameters 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
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
Movement Primitive (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 V: Hyperparameters 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
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 VI: Hyperparameters 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
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