TRAJDELETER: Enabling Trajectory Forgetting in Offline Reinforcement Learning Agents

Chen Gong*, Kecen Li[†][‡], Jin Yao*, and Tianhao Wang*
*University of Virginia
[†]Chinese Academy of Sciences

***Abstract*—Reinforcement learning (RL) trains an agent from experiences interacting with the environment. In scenarios where online interactions are impractical, offline RL, which trains the agent using pre-collected datasets, has become popular. While this new paradigm presents remarkable effectiveness across various real-world domains, like healthcare and energy management, there is a growing demand to enable agents to rapidly and completely eliminate the influence of specific trajectories from both the training dataset and the trained agents. To meet this problem, this paper advocates TRAJDELETER, the first practical approach to trajectory unlearning for offline RL agents. The key idea of TRAJDELETER is to guide the agent to demonstrate deteriorating performance when it encounters states associated with unlearning trajectories. Simultaneously, it ensures the agent maintains its original performance level when facing other remaining trajectories. Additionally, we introduce TRAJAUDITOR, a simple yet efficient method to evaluate whether TRAJDELETER successfully eliminates the specific trajectories of influence from the offline RL agent. Extensive experiments conducted on six offline RL algorithms and three tasks demonstrate that TRAJDELETER requires only about 1.5% of the time needed for retraining from scratch. It effectively unlearns an average of 94.8% of the targeted trajectories yet still performs well in actual environment interactions after unlearning. The replication package and agent parameters are available online[1].**

I. INTRODUCTION

Reinforcement learning (RL) develops agents to learn from trajectories (sequences of states, actions, and rewards experienced by an agent as it interacts with the environment over time) and has recently made significant strides in various complex decision-making areas, including robotics control [47], [27], recommendation systems [63], [73], and dialogue systems [62], [56], etc. In safety-critical areas like healthcare [46], [15], and even nuclear fusion [12], direct interaction with the environment can be hazardous, since a partially trained agent might cause damage. Thus, researchers have developed offline RL, a methodology where agents are trained using static datasets pre-collected from experts, manually programmed controllers, or even random strategies [17]. Offline RL paves the way for its application in situations where online interactions are either impractical or risky, and works well on a wide range of real-world fields [15], [71], [72], [28].

[‡]Work done as a remote intern at UVA.
[1]https://github.com/2019ChenGong/TrajDeleter/

Network and Distributed System Security (NDSS) Symposium 2025
24-28 February 2025, San Diego, CA, USA
ISBN 979-8-9894372-8-3
https://dx.doi.org/10.14722/ndss.2025.230126
www.ndss-symposium.org

With the success of offline RL comes the demand to delete parts of the training sets (also referred to as machine unlearning) for various reasons. For example, legislations like the European Union's General Data Protection Regulation (GDPR) [20] and the California Consumer Privacy Act (CCPA) [39] empowered users with the right to request their data to be deleted. The server may want to delete some data due to security or copyright reasons (the server discovered some data is poisoned, or copyrighted) [26]. This inspires the development of an "unlearning" methodology tailored for offline RL, which we term *offline reinforcement unlearning*.

Existing Solutions. A naive approach to unlearning is retraining without the data required to be unlearned. Furthermore, one can partition the training data and train an ensemble of models, so during unlearning, one still retrain a model but on a partition of the training set [5]. The inefficiency of retraining drives the development of approximate unlearning (but focuses more on the supervised learning in the image or text domain), including gradient ascent [64], [69] and contrastive learning [74]. We discuss unlearning methods in other fields in Section IX.

The unique paradigm of RL poses challenges when applying existing approximate unlearning methods. Specifically, the data in RL are a sequence of *trajectories*, each in the format of a tuple of state, action, and reward. Ye et al. first proposed the concept of reinforcement unlearning in the *online* setting by using environment poisoning attacks [70]. However, this is not feasible in offline RL. Our work is the first to address the need for unlearning within offline RL, specifically emphasizing unlearning at the trajectory level (Ye et al. [70] works at the aggregated level). Trajectory-level studies [57], [13] have attracted more attention, as RL algorithms are often trained within a single environment [50].

Our Proposal. This paper introduces TRAJDELETER, enabling offline RL agents to unlearn trajectories. TRAJDELETER is composed of two phases, "forgetting" and "convergence training". The forgetting phase first minimizes the value function Q (a function specific to RL to be described in Section II-A) for the unlearning samples. We choose to work with Q because it estimates the expected cumulative reward an agent can achieve from a state. So if the agent's cumulative reward for a state is low, it means the agent is unfamiliar with a trajectory, thus forgetting. However, this process alone will make the agent unstable for the normal, remaining samples. To alleviate this, we also maximize Q on remaining samples simultaneously, balancing unlearning and preventing performance degradation.

Due to the notorious unstable training problem in RL [57], [50], [45], which presents that the challenges encountered when the learning process of an agent does not progress consistently, resulting in erratic performance, slow convergence, or failure to learn an optimal policy. Our strategy of achieving the two opposite directions of optimization on the unlearning and remaining dataset may fail to guarantee the convergence of the unlearned agent, leading to potential instability during the training. To mitigate this concern, the second "convergence training" phase minimizes the discrepancies in cumulative rewards obtained by following the original and unlearned agents when encountering states in other remaining trajectories. Theoretical analysis presents that fine-tuning the unlearned agent ensures its convergence.

It is also crucial to evaluate if the trajectories with specific impacts have been erased from the approximate unlearned agent. This evaluation forms the fundamental basis of offline reinforcement unlearning. Du et al. [13] proposed ORL-AUDITOR to audit trajectories for offline DRL models, providing the potential tool for evaluating unlearning. However, ORL-AUDITOR can be time-consuming, owing to the extensive training required for numerous *shadow agents*, which explicitly exclude the unlearning trajectories from their training set. These agents are used to simulate or mimic the behavior of a target agent to distinguish between members and non-members of the training set. Specifically, to expedite the auditing process, we introduce TRAJAUDITOR, which fine-tunes the original agent (needed for unlearning) to create the shadow agents. We consider these shadow agents to be trained with datasets that include the targeted unlearning trajectories. In addition, we implement state perturbations along the trajectories, producing diverse auditing bases. Referring to Du et al. proposed [13], TRAJAUDITOR determines the success of unlearning by comparing cumulative rewards from unlearning trajectories. It assesses the similarity between results from shadow agents and the unlearned agent, with low similarity indicating successful unlearning. TRAJAUDITOR matches the performance of ORL-AUDITOR while requiring significantly fewer computing resources.

Evaluations. We extensively experiment with six offline RL algorithms on three common Mujoco evaluation tasks [60] to verify the effectiveness and efficiency of TRAJDELETER. We experimented with various unlearning rates (the proportion of data required to be forgotten in a dataset). TRAJAUDITOR shows high proficiency, achieving average F1-scores of 0.88, 0.87, and 0.88 across three tasks. It achieves a 97.8% reduction in time costs while attaining an F1-score that is 0.08 points higher compared to training shadow agents from scratch in ORL-AUDITOR. It is a simple yet efficient tool for determining whether a specific trajectory continues influencing the unlearned agent, paving the path for our unlearning study.

Then, we evaluate how effective is TRAJDELETER under the assessment of TRAJAUDITOR. Our experiments show that TRAJAUDITOR achieves the removal of 92.7%, 99.5%, and 90.5% of targeted trajectories across three tasks while requiring only 1.5% of the time needed for retraining from scratch. The average cumulative returns show a slight difference of 2.2%, 0.9%, and 1.6% between TRAJDELETER-unlearned agents and retrained agents in the three tasks on average.

We also analyze hyper-parameters of TRAJDELETER. With an increase in the number of forgetting steps, the performance of TRAJDELETER significantly improves, resulting in the unlearned agent forgetting more trajectories. We also observe an interesting fact: with an increased number of forgetting steps, TRAJDELETER shows minimal sensitivity to the values of other hyper-parameters it introduces. We also conduct ablation studies to study the significance of "convergence training" in enhancing TRAJDELETER's efficacy, and investigate its effectiveness in defending against trajectory poisoning attacks.

Contributions. In summary, our contributions are three-fold:

- To our knowledge, we propose the first practical trajectories-level unlearning approach, TRAJDELETER, specifically tailored for offline RL agents.
- We introduce TRAJAUDITOR, a simple yet efficient method to assess whether the unlearning method effectively erases the specific trajectories' influence on the unlearned agent.
- We perform a comprehensive evaluation of TRAJDELETER. The results present the effectiveness of TRAJDELETER in offline DRL agents trained across six distinct prevalent algorithms and three tasks.

We recognize that approximate unlearning's compatibility with legal requirements is uncertain, but the similar situation, where technology moves faster than the legal side, is true in other areas, like differential privacy [14] and watermark [41]. Approximate unlearning retains significant research value.

II. BACKGROUND

A. *Offline Reinforcement Learning*

The training of DRL agents operates a trial-and-error paradigm, with learning driven by feedback from rewards. For example, we consider an agent responsible for controlling a car. If it accelerates when confronted with a red traffic light, it receives a penalty in the form of a negative reward. Then, this agent will update its policy to avoid accelerating when a traffic light turns red. The agent learns from such experiences by interacting with the environment during the training phase, which is called the *online* RL.

However, the online settings are not always feasible. For instance, a hospital aims to train an RL agent to recommend treatments to future patients. The online RL would make the agent propose treatments, observe the results, and adjust its policy accordingly. Since it is ethically and practically problematic to experiment with patients' health, the offline RL, which is designed to train agents from a *pre-collected* and *static* dataset eliminating the need for interactions with real environments during the training stages, is more suitable. We highlight that online RL trains an agent in real-time through continuous interaction with the environment, updating its policy based on received feedback (rewards or penalties). In contrast, offline RL trains an agent using a fixed, pre-collected dataset without accessing the environment during the training phase. Then, as presented in Figure 1, we outline the three-step implementation process of offline RL, including data collection, training the offline RL agent, and deployment.

Data Collection. At timestep t, an agent observes a state s_t and executes an action a_t determined by the agent's *policy*. The agent selects an action a_t according to the policy $\pi(\cdot)$, which dictates which actions to execute in a given state s_t, i.e., $a_t \sim \pi(\cdot|s_t)$. After taking an action, the agent obtains an immediate reward from the environmental reward function, denoted as $r_t = \mathcal{R}(s_t, a_t)$. Then, the environment transitions to a new state s_{t+1}, determined by the transition function $s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)$. This sequence persists until the agent reaches a termination state s_T. This process results in a *trajectory*,

$$\tau : (s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, \cdots, s_{|\tau|}, a_{|\tau|}, r_{|\tau|}).$$

Besides, the four-tuple $\langle s_t, a_t, r_t, s_{t+1} \rangle$ in each trajectory is referred to as the *transition*. The organizer, also named the data provider (e.g., the hospital), gathers trajectories through multiple, distinct policies interacting with the environments. Then, these trajectories consist of the offline dataset $\mathcal{D} = \{\tau^i\}_{i=1}^N$, where N represents the total number of trajectories.

Training. In online RL, an agent aims to learn an optimal policy π^* that can get high expected performance from the environment. Specifically, the *cumulative discounted return* is the sum of discounted rewards within a given trajectory: $R(\tau) = \sum_{i=0}^{T} \gamma^i r_i$, where $\gamma \in (0, 1)$ represents the discount factor [57] and T is the length of the trajectory. Hence, the objective of online RL can be formulated to optimize the policy and achieve the highest possible cumulative discounted return, which is defined as,

$$\pi^* = \arg\max_{\pi} \mathbb{E}_{\tau\sim\pi} \left[R(\tau^\pi) \right], \tag{1}$$

where τ^π indicates the trajectory generated by the policy π. The solution to Eq. (1) could be accomplished by maximizing *action-state value function* $Q^\pi(s, a)$ [57], which is defined as,

$$Q^\pi(s, a) = \mathbb{E}_{\tau\sim\pi} \left[R(\tau^\pi) | s_0 = s, a_0 = a \right]. \tag{2}$$

$Q^\pi(s, a)$ quantifies the expected cumulative reward an agent can achieve starting from a state s, taking action a, and following a policy π. In other words, it serves as a metric for how good it is for an agent to execute action a while being in state s. Thus, the objective in Eq. (1) can be reformulated as, $\pi^* = \arg\max_\pi \mathbb{E}_{a\sim\pi} \left[Q^\pi(s, a) \right], \forall s \in \mathcal{S}$, where $\mathcal{S}$ refers to the state space. Next, we explain how to optimize the policy in offline RL settings.

In offline RL, the agent is only allowed to learn from the trajectories present in the offline dataset, which is defined as $\mathcal{D}$. The agent cannot access states absent from the given offline

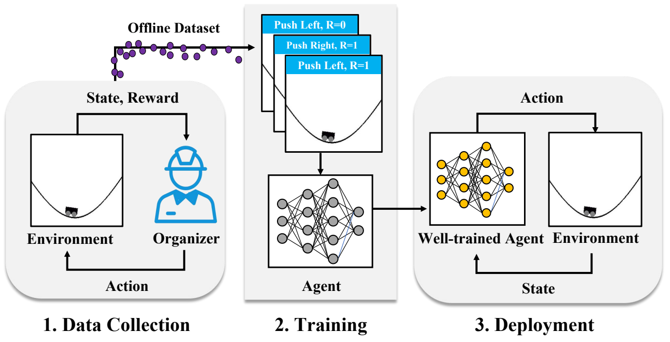

Fig. 1: An example of offline RL implementations. Initially, the organizer gathers trajectories through interactions with the environments, forming the offline dataset. Then, the agent is trained using this static dataset. Once fully trained, the agent is deployed in real-world applications.

dataset. Thus, distinguished from considering all states in state space, offline RL only takes into account the states found within the offline dataset. In particular, offline RL initially requires the agent to derive an understanding of the dynamical system underlying the environment entirely from the pre-collected offline dataset. Then, it needs to establish a policy $\pi(a|s)$ that maximizes possible performance when *actually used to interact with the environment* [40].

Deployment. The well-trained offline agents are prepared for deployment. Their deployment ensures that they can make decisions based on the rich trajectories derived from the datasets, ensuring efficiency and safety in real-time applications.

B. Machine Unlearning

In response to the demands of recently introduced legislation, like the European Union's General Data Protection Regulation (GDPR) [20], a new branch of privacy-preserving machine learning arises, known as *machine unlearning*. This concept requires that the specified training data points and their influence can be erased completely and quickly from both the training dataset and trained model [5]. Specifically, we utilize the algorithm to train a model on a dataset. The well-trained model performs certain functions, such as classification, regression, and more. Upon receiving requests for the model to "forget" a subset of the training dataset, the unlearning algorithm is capable of altering the trained model so that it behaves as if it is trained solely on the remaining dataset (excluding the subset that is required to be forgotten).

With the rapid progress in applying offline RL in reality [71], [72], [55], [46], this paper focuses on the unlearning requirement in the field of offline RL. We propose TRAJDELETER, aiming to "unlearn" selected training trajectories by updating the trained agent to completely eliminate the influence of these trajectories from the updated agent.

C. Unlearning Scenarios

We outline the key requirements for advancing offline reinforcement unlearning, highlighting three typical scenarios.

Privacy Concerns. Unlearning has become especially relevant in privacy and data protection laws, including the European

Union's GDPR [20], which enforces a "right to erasure." This right also should be involved in offline RL. For instance, we use the entire offline dataset to train an agent that performs brilliantly in real-world tasks. However, there may be instances where institutions need to instruct these trained agents to "forget" certain trajectories containing sensitive information, like credit card details or confidential communications. Given the potential for privacy attacks on training data from agents [48], [24], [68], implementing trajectory unlearning becomes crucial to mitigate these concerns.

Trajectory Poisoning. We focus on poisoning attacks within the RL area [16], [65], [44], where an attacker aims to mislead the trained agent by editing the training trajectory. This intervention greatly reduces the performance of the learning agent. We implement unlearning to restore the agent's performance efficiently, avoiding the time-consuming retraining.

Copyright Issue. Numerous studies indicate that our dataset may be susceptible to various forms of misuse, including infringements of intellectual property rights [4], [58]. The copyright issue is particularly pressing, given the increasing prevalence of sophisticated data utilization across domains. Our method enables the rapid and efficient removal of training trajectories that lack a clearly defined copyright source. This method helps mitigate legal risks associated with copyright violations and ensures compliance with evolving copyright laws and ethical standards in data usage.

III. Problem Setup and Preliminaries

A. Formal Problem Definition

We first introduce the concept of offline reinforcement unlearning and the formal framework for this problem. We assume the training dataset consists of N trajectories and formally express it as $\mathcal{D} = \left\{\tau^i | \tau^i = \langle s_t^i, a_t^i, r_t^i, s_{t+1}^i \rangle_{t=0}^{|\tau^i|}\right\}_{i=1}^{N}$. Then, we define an offline RL algorithm as a function $\mathcal{A}(\cdot) : \mathcal{D} \mapsto \Pi$, which maps a dataset $\mathcal{D}$ to a trained agent within the hypothesis space Π. Then, we use the notion $\mathcal{D}_f = \left\{\tau^i\right\}_{i=1}^{M}$ $(M < N)$ to represent the subset $\mathcal{D}_f \subset \mathcal{D}$, which the agent is required to forget. Besides, the modified dataset is defined as $\mathcal{D}_m = \mathcal{D} \backslash \mathcal{D}_f$, indicating the portion of the dataset that we intend for the agent to retain. The offline reinforcement unlearning method $\mathcal{U}$: $\mathcal{A}(\mathcal{D}) \times \mathcal{D} \times \mathcal{D}_f \mapsto \Pi$, indicates a function that maps an agent $\mathcal{A}(\mathcal{D})$, along with the training dataset $\mathcal{D}$, and a subset $\mathcal{D}_f$ designated for removal, to a correspondingly unlearned agent within the hypothesis space Π. The offline reinforcement unlearning process, $\mathcal{U}(\mathcal{A}(D), \mathcal{D}, \mathcal{D}_f)$, is defined as function that takes a trained agent $\pi \leftarrow \mathcal{A}(\mathcal{D})$, a training dataset $\mathcal{D}$, and a dataset $\mathcal{D}_f$ that should be forgotten. This process aims to guarantee that the unlearned agent: $\pi' \leftarrow \mathcal{U}(\mathcal{A}(D), \mathcal{D}, \mathcal{D}_f)$ behaves as an agent directly trained on the training dataset excluding $\mathcal{D}_f$.

B. Challenges

This section delves into discussing three distinct challenges associated with offline reinforcement unlearning.

- **How can we evaluate the efficacy of offline reinforcement unlearning?** We focus on approximate unlearning, and should essentially assess whether a trajectory is part of the agent's training dataset. One natural approach is to leverage membership inference attacks (MIAs), and there have been MIAs against DRL [48], [24], [70]. While most of these are aimed at online RL, Du et al. [13] introduced ORL-AUDITOR to audit datasets for offline DRL models. However, their process is time-consuming due to the extensive training of numerous shadow agents. A simple yet effective method for evaluating approximate offline reinforcement unlearning is the fundamental basis of our study.
- **How can we unlearn trajectories from the agent's policy?** The objective of offline reinforcement unlearning is to eliminate the trajectories' impact on the agent, essentially to "forget trajectories." Currently, there is a lack of established methods for unlearning trajectories in the field of offline RL. Therefore, the primary challenge lies in devising an effective unlearning methodology tailored for offline RL.
- **How can we prevent degradation of performance after unlearning?** As the training of RL especially suffering from unstable training [2], [17], the unlearning process can result in performance degradation. Ensuring that the unlearned agent retains its effectiveness poses a significant challenge.

C. Our Proposals

To address the first challenge, we propose TRAJAUDITOR. Contrasting with the approach in ORL-AUDITOR [13], which involves training shadow agents from scratch, TRAJAUDITOR adopts a more direct method by fine-tuning the original agent to generate the shadow agents. Besides, we introduce perturbations to the states within the trajectories to generate diverse bases for auditing. These two processes significantly reduce the time required for auditing.

To overcome the second challenge, referring to the proposed definition of "forgetting an environment" in Ye et al. [70], we interpret "forgetting trajectories" as the agent demonstrating reduced or deteriorating performance when encountering states involved in those trajectories. This interpretation is in line with intuitive understanding. When an agent has learned a trajectory and is familiar with its states, it can perform more effectively upon encountering similar states. This results in improved performance, as the agent is better equipped to execute optimal actions in these familiar states. Alternatively, when the agent encounters unfamiliar states from forgotten trajectories, it tends to make sub-optimal decisions.

To address the third challenge, we suggest fine-tuning the unlearned agent to minimize the disparity between the value functions of the unlearned and the original agents. Specifically, this approach aims to synchronize the value function of state-action pairs in the remaining offline dataset.

IV. TRAJAUDITOR

A. Existing Solutions

The foundational metric of our study is to determine if a trajectory is included in the agent's training dataset, which is

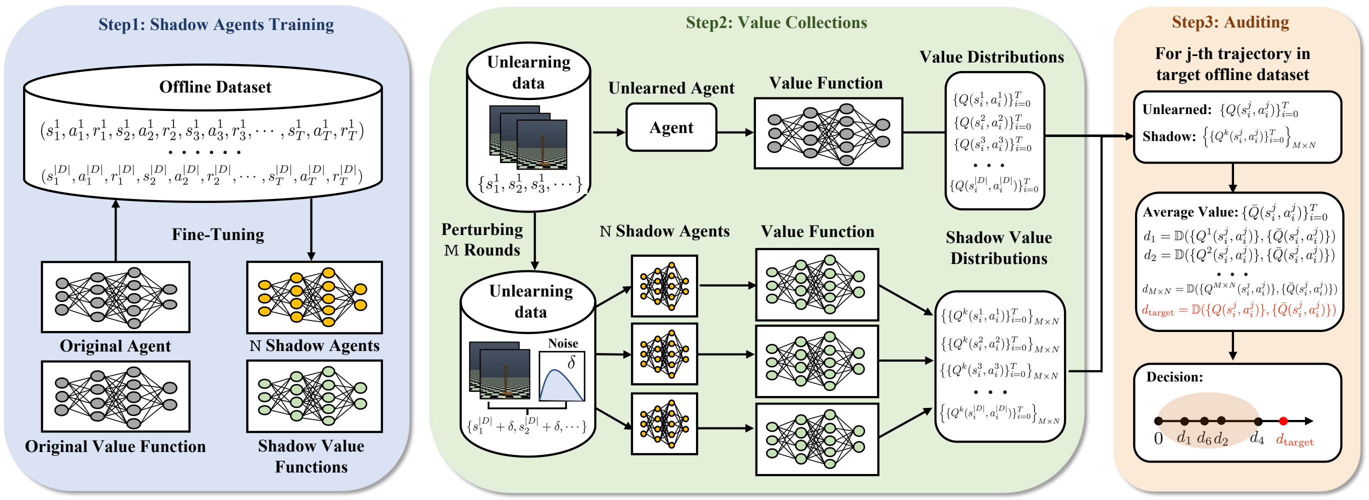
Step1: Shadow Agents Training
Offline Dataset
$(s_1^1, a_1^1, r_1^1, s_2^1, a_2^1, r_2^1, s_3^1, a_3^1, r_3^1, \cdots, s_T^1, a_T^1, r_T^1)$
$(s_1^{|D|}, a_1^{|D|}, r_1^{|D|}, s_2^{|D|}, a_2^{|D|}, r_2^{|D|}, \cdots, s_T^{|D|}, a_T^{|D|}, r_T^{|D|})$
Fine-Tuning
Original Agent
N Shadow Agents
Original Value Function
Shadow Value Functions
Step2: Value Collections
Unlearning data
$\{s_1^1, s_2^1, s_3^1, \cdots\}$
Unlearned Agent
Agent
Value Function
Value Distributions
$\{Q(s_i^1, a_i^1)\}_{i=0}^T$
$\{Q(s_i^2, a_i^2)\}_{i=0}^T$
$\{Q(s_i^3, a_i^3)\}_{i=0}^T$
$\{Q(s_i^{|D|}, a_i^{|D|})\}_{i=0}^T$
Perturbing M Rounds
N Shadow Agents
Value Function
Shadow Value Distributions
Unlearning data
Noise
δ
$\{s_1^{|D|}+\delta, s_2^{|D|}+\delta, \cdots\}$
$\{\{Q^k(s_i^1, a_i^1)\}_{i=0}^T\}_{M\times N}$
$\{\{Q^k(s_i^2, a_i^2)\}_{i=0}^T\}_{M\times N}$
$\{\{Q^k(s_i^3, a_i^3)\}_{i=0}^T\}_{M\times N}$
$\{\{Q^k(s_i^{|D|}, a_i^{|D|})\}_{i=0}^T\}_{M\times N}$
Step3: Auditing
For j-th trajectory in target offline dataset
Unlearned: $\{Q(s_i^j, a_i^j)\}_{i=0}^T$
Shadow: $\{\{Q^k(s_i^j, a_i^j)\}_{i=0}^T\}_{M\times N}$
Average Value: $\{\bar{Q}(s_i^j, a_i^j)\}_{i=0}^T$
$d_1 = \mathbb{D}(\{Q^1(s_i^j, a_i^j)\}, \{\bar{Q}(s_i^j, a_i^j)\})$
$d_2 = \mathbb{D}(\{Q^2(s_i^j, a_i^j)\}, \{\bar{Q}(s_i^j, a_i^j)\})$
$d_{M\times N} = \mathbb{D}(\{Q^{M\times N}(s_i^j, a_i^j)\}, \{\bar{Q}(s_i^j, a_i^j)\})$
$d_{\text{target}} = \mathbb{D}(\{Q(s_i^j, a_i^j)\}, \{\bar{Q}(s_i^j, a_i^j)\})$
Decision:
0
d_1
d_6
d_2
d_4
d_{target}

Fig. 2: The workflow of TRAJAUDITOR. "Shadow Agents Training" fine-tunes the original target agent under investigation, and its value function, to gather N shadow agents and shadow value functions. "Value Collections" calculates the value (as described in Eq. (2)) distributions of the states of target trajectories from the unlearning agents, i.e., $\left\{Q\left(s_i^{|D|}, a_i^{|D|}\right)\right\}_{i=0}^{T}$, where T is the length of trajectories. This step introduces noise, denoted as δ, to the states M times, calculating the value distributions of perturbed trajectories via shadow models. Thus, for each trajectory, we obtain $M \times N$ value distributions. The "Auditing" assesses the distances between the average value distribution and the value distributions of both the unlearned agent (d_{target}) and the shadow agents ($d_{M\times N}$). These similarities determine whether trajectories belong to the training dataset.

crucial for understanding approximate unlearning. Being the first to introduce offline reinforcement unlearning, we recognize the absence of a definitive method to quantify this. In another field of approximate unlearning, MIAs are widely used for unlearning methods' evaluation [66], [74]. This motivates us to consider using MIAs to evaluate offline reinforcement learning as well. In supervised learning, loss is an effective metric for evaluating the effectiveness of unlearning [54], [8]. However, as our experiments detailed in Appendix D1, the loss difference between trained and non-trained trajectories is minimal, rendering it unsuitable as a basis for auditing. MIAs in online RL [48], [24], [70] assume that attackers have control over environments, enabling them to gather trajectories and manipulate them. However, in offline RL, such operations are impractical as we cannot access the environments. Du et al. [13] emphasize the same perspective that MIA applicable in online RL is not applicable to offline RL.

Du et al. [13] introduced ORL-AUDITOR for auditing datasets in offline RL models. This approach leverages the concept that *cumulative rewards* can serve as a unique identifier. In particular, Q^π measures the expected cumulative reward that an agent can attain, beginning from state s, executing action a, and adhering to policy π. We consider the series of states $\left\{(s_t)_t^T\right\}$ within a trajectory. Feeding this series of states into a tested agent π generates a sequence of state-action pairs $\left\{(s_t, \pi(s_t))_t^T\right\}$. Subsequently, by evaluating this sequence of state-action pairs with Q^π, we can derive a *value vector* $\vec{Q}^\pi = \left\{Q^\pi(s_t, \pi(s_t))_t^T\right\}$ for a trajectory and the agent. We assemble a collection of shadow agents with the assurance that their training datasets incorporate the target trajectories. The basis for our audit relies on the similarity between the value vector of the target trajectories generated by these shadow agents and that generated by the tested agent.

B. Our Method

Different from MIAs, in evaluation, we can involve the target model before and after unlearning. Therefore, we do not need to train the shadow models from scratch. It suffices to fine-tune the target original model to obtain the shadow models. It is noticed that TRAJAUDITOR is not a type of MIA. In TRAJAUDITOR, we should involve a model that we confirm includes the target trajectory in its training dataset, which is unpractical in MIAs. Compared to training many shadow agents from scratch, fine-tuning significantly saves computational overheads. Moreover, to ensure a robust estimation from shadow models, we use state perturbation to ensure shadow models cover a wide range of possibilities.

We call our method TRAJAUDITOR and plot its main steps in Figure 2. We elaborate on the technical details of TRAJAUDITOR as follows.

Shadow Model Preparation. We can define that the unlearning dataset $\mathcal{D}_f$ is included in the training dataset of the original agent π. We then directly fine-tune the original agent π on the entire dataset $\mathcal{D}$ to gather a set of shadow agents $\{\pi_i^s\}_{i=0}^N$, where N is the number of shadow agents. *We believe these shadow agents have been thoroughly trained on the unlearning dataset* $\mathcal{D}_f$. Besides, as outlined in Section II-A, the fine-tuning process also necessitates concurrently updating the value function, allowing us to obtain the value functions of the shadow agents $\left\{Q^{\pi_i^s}\right\}_{i=0}^N$.

Value Collections. For jth ($j = 1, \cdots, |\mathcal{D}_f|$) trajectory in the unlearning dataset $\mathcal{D}_f$, we query the unlearned agent's value vector $\vec{Q}_j^{\pi'} = Q^{\pi'}\left(s_t^j, \pi'\left(s_t^j\right)\right), (t = 1, \cdots, T)$.

Additionally, for the j_{th} trajectory, we collect the value vectors $\vec{Q}_j^{\pi_i^s} = Q^{\pi_i^s}\left(s_t^j, \pi_i^s\left(s_t^j\right)\right), (t = 1, \cdots, T;\ i = 1, \cdots, N)$ generated by shadow agents. Besides, we add a low-level noise δ to perturb states to generate more value vectors for a trajectory, further reducing the requirement for extensive shadow agent training. By using perturbations across M rounds $\left\{(s_t^j + \delta_h)_{h=0}^M\right\}$ and utilizing N shadow agents, we gather $M \times N$ value vectors for each trajectory within the unlearning dataset, which is recorded as $\vec{Q}_{j,m}^{\pi_i^s} (i = 1, \cdots, N;\ j = 1, \cdots, M)$.

Auditor. This step relies on the similarity between the value vector of the unlearning trajectories generated by shadow agents and that generated by the unlearned agent. Specifically, for j_{th} trajectory, we define the mean of $\left\{\vec{Q}_{j,m}^{\pi_i^s}\right\}$ as $\vec{Q}_{j,\text{ave}}^{\pi^s}$. Referring to the practical adoption in ORL-AUDITOR [13], we conduct Grubbs' hypothesis test [31] to ascertain the success of removal. This determination is made when $d_{\text{target}} = \mathbb{D}\left(\vec{Q}_{j,\text{ave}}^{\pi^s}, \vec{Q}_j^{\pi'}\right)$ falls outside the distribution of $\left\{\mathbb{D}\left(\vec{Q}_{j,\text{ave}}^{\pi^s}, \vec{Q}_{j,m}^{\pi_i^s}\right) \middle| i \in \{1, \cdots, N\}, j \in \{1, \cdots, M\}\right\}$, where $\mathbb{D}$ denotes the Wasserstein distance [49] between two vectors. Otherwise, TRAJAUDITOR determines that the trajectories, which are required to be forgotten, continue to influence the unlearned agent π'.

V. TRAJDELETER

A. Strawman Unlearning Methods

Ye et al. proposed "environment unlearning" in online RL. We cannot directly apply reinforcement environment unlearning for online RL [70] because it requires the user to poison the transition function $\mathcal{T}(\cdot|s_t, a_t)$ of the environment (the training for offline RL is restricted from accessing environments).

Another baseline is to perturb the rewards (it has been understood that agents learn via rewards [57]; thus, perturbing rewards prevents agents from learning the corresponding behaviors). We modify the rewards of unlearned trajectories by assigning random rewards sampled from the uniform distribution. Then, we fine-tune the agent using this modified dataset for unlearning. This approach constitutes the "random-reward" baseline in Section VI-B. Moreover, we can also assign unlearned trajectories with small or the worst possible rewards, but they could adversely affect the agent's performance.

To overcome this issue, we propose to enhance the agent's performance on the remaining dataset in the unlearning stage, which is crucial to prevent the degradation of the agents' performance. The unlearning approach now seems to be successful. However, due to the unstable training in RL [17], [38], [19], the methods mentioned may be susceptible to the issue of policy divergence (as the empirical analysis presented in Section VIII-A). Therefore, we need to fine-tune the agents on the remaining dataset to encourage policy convergence.

B. Overview

As discussed in Section III-A, we aim to update the optimal unlearned policy to ensure the agent that *takes actions of lower value in unlearning trajectories while still preserving its performance in the rest of the trajectories*. To achieve this goal, we structure TRAJDELETER into two distinct components that align with the first and second terms of Eq. (3) as follows: (1) *Forgetting*, and (2) *Convergence Training*.

$$\begin{cases} \min \mathbb{E}_{s\sim\mathcal{D}_f}\left[Q^{\pi'}(s, \pi'(s))\right] + \max \mathbb{E}_{s\sim\mathcal{D}_m}\left[Q^{\pi'}(s, \pi'(s))\right] \\ \min \mathbb{E}_{(s,a)\sim\mathcal{D}_m}\left[\left\|Q^{\pi'}(s,a) - Q^{\pi}(s,a)\right\|_\infty\right]. \end{cases} \tag{3}$$

The first term of Eq. (3) represents the 'forgetting' phase, instructing the learned policy π' to take suboptimal actions in the unlearning dataset $\mathcal{D}_f$, while maintaining its normal behavior in the remaining dataset $\mathcal{D}_m$. We achieve this by updating the agent to deliberately minimize its cumulative reward in the states belonging to the unlearning trajectories, thereby steering it towards making less effective decisions in those states. Consequently, we minimize the value function for states within the unlearning trajectories, aligning the agent's actions with the unlearning objectives. Training an agent by solely minimizing the value function could lead to the issue of agent collapse – the agent's performance tends to decline rapidly when interacting with the environment [26]. To mitigate this problem, we also focus on maximizing the agent's value for states in the remaining dataset, thereby balancing the unlearning process and preventing the agent from deteriorating.

The second term of Eq. (3), aligning with the "convergence training" phase, aims to reduce the value differences between the original and the unlearned agent on the remaining dataset $\mathcal{D}_m$. This step ensures we can fine-tune the unlearned agent's convergence, as discussed in Section V-D.

C. Technical Details

This section describes how TRAJDELETER enables offline RL agents to forget specific trajectories. We outline the processes for "forgetting" and "convergence training," as follows.

Forgetting. This learning phase trains the agent to forget specific trajectories while maintaining effectiveness in other trajectories. We initially reformulate the objective function as,

$$\mathcal{L}_1 = \max \mathbb{E}_{s\sim\mathcal{D}_m}\left[Q^{\pi'}(s, \pi'(s))\right] - \mathbb{E}_{s\sim\mathcal{D}_f}\left[Q^{\pi'}(s, \pi'(s))\right].$$

Our unlearning process begins with the original policy π (the agent required being unlearned). We use a neural network represented by π'_θ to denote the policy after unlearning. Our objective is to optimize the parameter θ to maximize the value function, which is given by,

$$\mathbb{E}_{s\sim\mathcal{D}_m, a\sim\pi'_\theta}\left[Q^{\pi'_\theta}(s,a)\right] - \lambda\mathbb{E}_{s\sim\mathcal{D}_f, a\sim\pi'_\theta}\left[Q^{\pi'_\theta}(s,a)\right], \tag{4}$$

where λ is a constant to balance the unlearning process and prevent the agent from deteriorating. We then compute the gradient of the objective function with respect to the parameters and iteratively apply stochastic gradient-ascend to approach a local maximum in $\mathcal{L}_1(\theta)$. Specifically, at iteration k, we update the policy by gradient ascent, $\theta_{k+1} \leftarrow \theta_k + \nabla_\theta \mathcal{L}_1(\theta)|_{\theta_k}$. Based

Algorithm 1: Workflow of TrajDeleter

```
1  Input: $\pi$: The original agent, parameterized by $\theta_o$. $\pi'_\theta$:
       The unlearning policy, parameterized by $\theta$. Value
       Functions: $Q^\pi$ and $Q^{\pi'}$, with parameters $\phi_o$ and $\phi$,
       respectively. $\mathcal{D}$: The complete offline dataset. $\mathcal{D}_f$:
       The dataset the agent needs to forget. $\mathcal{D}_m$: The
       remaining dataset post unlearning.
2  Initialization: initialize $\theta^{(0)} = \theta_o$, $\phi^{(0)} = \phi_o$.
   // Forgetting
3  for $k = 1, 2, 3, \cdots, K$ do
4      Sample trajectories from $\mathcal{D}_m$ and $\mathcal{D}_f$, and collect
         a batch of trajectories $\mathcal{D}_{Bm} = \{\tau_i\}$, $\mathcal{D}_{Bf} = \{\tau_j\}$,
         where $i, j = 1, 2, \cdots, B$.
5      For each trajectory $\tau_i$ and $\tau_j$, compute the
         advantage at each time step $t$ of trajectories:
         $A^{i_t}_{\pi'}(s^i_t, a^i_t) = Q^{\pi'}\left(s^i_t, a^i_t\right) - \mathbb{E}_{a\sim\pi'}\left[Q^{\pi'}\left(s^i_t, a\right)\right]$,
         $A^{j_t}_{\pi'}(s^j_t, a^j_t) = Q^{\pi'}\left(s^j_t, a^j_t\right) - \mathbb{E}_{\pi'}\left[Q^{\pi'}\left(s^j_t, a\right)\right]$.
6      We have $A^{i_{0:|\tau_i|}}_{\pi'}$ and $A^{j_{0:|\tau_j|}}_{\pi'}$ $(i, j = 1, \cdots, B)$ in
         Eq. (5). Then, we can obtain the policy gradient
         to maximize the Eq. (4) by updating $\theta^{(k)}$.
7      Update $\phi^{(k)}$ by minimizing the loss of Eq. (6).
8  end
   // Convergence training
9  for $h = 1, 2, 3, \cdots, H$ do
10     Sample trajectories from $\mathcal{D}_m$, and collect a batch
         of trajectories $\mathcal{D}_B = \{\tau_i\}$, where $i = 1, \cdots, B$.
11     For each trajectory $\tau_i$, compute the advantage at
         each time step $t$ of trajectories:
         $A^{i_t}_{\pi'}(s^i_t, a^i_t) = Q^{\pi'}\left(s^i_t, a^i_t\right) - \mathbb{E}_{a\sim\pi'}\left[Q^{\pi'}\left(s^i_t, a\right)\right]$.
12     We have $A^{i_{0:|\tau_i|}}_{\pi'}$ $(i = 1, \cdots, B)$ in Eq. (8) and can
         obtain the policy gradient to update $\theta^{(K+h)}$.
13     Update $\phi^{(K+h)}$ by minimizing the loss of Eq. (7)
         and Eq. (6) on the $\mathcal{D}_{Bm}$.
14 end
15 Output: $\pi'_\theta$: the well-trained unlearned agent.
```

on the Policy Gradient Theorem [45], we express the policy gradient of policy π_θ as,

$$\begin{aligned}\nabla_{\theta_k}\mathcal{L}_1 = \mathbb{E}_{s\sim\mathcal{D}_m, a\sim\pi'_{\theta_k}}\left[\nabla_{\theta_k}\log\pi'_{\theta_k}(s,a)A_{\pi'_{\theta_k}}(s,a)\right] \\ - \lambda\mathbb{E}_{s\sim\mathcal{D}_f, a\sim\pi'_{\theta_k}}\left[\nabla_{\theta_k}\log\pi'_{\theta_k}(s,a)A_{\pi'_{\theta_k}}(s,a)\right].\end{aligned} \tag{5}$$

Here, $A_\pi(s,a)$ indicates the advantage function, measuring the difference between the value of state-action pair (s,a) and the average value of that state s. Using this advantage function, we can determine the improvement of taking a particular action in a given state over the average. In Eq. (5), the advantage function $A_{\pi'_{\theta_k}}(s,a)$ is defined as, $A_{\pi'_{\theta_k}}(s,a) = Q^{\pi'_{\theta_k}}(s,a) - \mathbb{E}_{a\sim\pi'_{\theta_k}}\left[Q^{\pi'_{\theta_k}}(s,a)\right]$. Essentially, this function computes the additional reward the agent receives from choosing that action [30].

To address the Eq. (5), we are required to approximate function $Q^{\pi'_{\theta_k}}$ for each iteration. We start this approximation by an neural network parameterized with ϕ_k to model $Q^{\pi'_{\theta_k}}(s,a)$ using the TD-learning paradigm [57], and then update this network for next state s' repeatedly by minimizing the TD-error $\mathcal{G}(\phi_k)$,

$$\mathbb{E}_{\mathcal{D}}\left\|Q^{\pi'_{\theta_k}}_{\phi_k}(s,a) - \left(r + \gamma\mathbb{E}_{a'\sim\pi_{\theta_k}}\left[Q^{\pi'_{\theta_k}}_{\phi_k}(s',a')\right]\right)\right\|_2, \tag{6}$$

where $(s,a,r,s') \sim \mathcal{D}$, and $\mathcal{D} = \mathcal{D}_m \cup \mathcal{D}_f$ represents the original dataset. By consistently updating the policy network and value function using the method mentioned above, the agent is effectively trained to "forget" the targeted trajectories.

Convergence training. Relying solely on this training might not ensure the convergence of π', potentially causing instability (as conducted experiments in Section VIII-A) in the unlearning process. To address this issue, we introduce "convergence training" in TrajDeleter, as depicted in the second term of Eq. (3). The stage focuses on minimizing the difference between the value functions of the unlearned policy π' and the fixed original policy π, aiming to align the value function of π'_θ closely with that of π and providing the convergence guarantee for TrajDeleter.

This "forgetting" phase yields a trained policy that serves as the "convergence training" starting point. At iteration h, we initially fine-tune the value function by,

$$\mathcal{L}_2(\theta) = \min\mathbb{E}_{(s,a)\sim\mathcal{D}_m}\left[\left\|Q^{\pi'_{\theta_h}}(s,a) - Q^\pi(s,a)\right\|_2\right]. \tag{7}$$

Then, analogous to the implication in Eq. (5), we update the policy π'_{θ_h} based on the Policy Gradient Theorem [45],

$$\nabla_{\theta_h}\mathcal{L}_2 = \mathbb{E}_{s\sim\mathcal{D}_m, a\sim\pi'_{\theta_h}}\left[\nabla_{\theta_h}\log\pi'_{\theta_h}(s,a)A_{\pi'_{\theta_h}}\right]. \tag{8}$$

After training, TrajDeleter fine-tunes the policy for convergence. We provide the theoretical analysis in Section V-D.

Summary. We outline TrajDeleter in Algorithm 1. This algorithm inputs the original agent and its value function Q^π and the dataset targeted for unlearning $\mathcal{D}_f$. The workflow involves a two-phase approach: first "forgetting" specific trajectories, then reinforcing the new policy through convergence training. In the forgetting phase, it iteratively processes batches of trajectories from both $\mathcal{D}_m$ and $\mathcal{D}_f$ (Line 4). The computation of advantages for each trajectory in these datasets (Line 5) involves updating the unlearning policy π'_θ (Line 6) and the value functions (Line 7) to gradually forget behaviors learned from $\mathcal{D}_f$. This phase is key in systematically erasing specific behaviors from the agent's learning.

Following this, we start the convergence training phase. Here, trajectories from $\mathcal{D}_m$ are sampled (Line 10). The advantage for each trajectory is computed similarly to the forgetting phase (Line 11). Then, updates are made to the policy and value function parameters to reinforce the unlearned agent (Line 12-13). This phase ensures that the agent's behavior post-unlearning remains consistent and effective. Upon completing the iterative process described, TrajDeleter produces a well-trained unlearned agent π'_θ.

D. Convergence Analysis

The "forgetting" phase yields a trained policy that serves as the start of the next stage. Then, the policy conducts fine-tuning guided by the second term of Eq. (3). We provide theoretical analysis to guarantee policy convergence after fine-tuning.

We assume that the original policy π remains fixed during training and approximate the optimal policy π^*, focusing solely on training the policy π' for unlearning. Theorem 1 states that as training progresses, the difference between the learned Q-function and the optimal Q-function diminishes.

Theorem 1 (Interaction convergence [61]): We assume that the offline dataset includes a diverse range of states. The state distribution generated by any policy is consistently bounded relative to the distribution in the offline dataset. Specifically, denoting the state distribution of the offline dataset as $\mu(s)$, for the state distribution $\nu(s)$ generated by any policy π_k, the condition $\forall s, \frac{\nu(s)}{\mu(s)} \leq C$ holds. Let Q^* indicate the optimal value function; we have,

$$\|Q^* - Q^{\pi_{k+1}}\|_\infty \leq \gamma \|Q^* - Q^{\pi_k}\|_\infty + \epsilon + C\|Q^{\pi_k}\|_\infty,$$

where π_k denotes a sequence of policies correlated to their respective value functions Q^{π_k}. Here, ϵ signifies the approximation error in value estimation: $\|Q^{\pi_k} - (r + \gamma Q^{\pi_{k+1}})\|_\infty$.

This formulation implies the convergence of value function Q towards the optimal value function with a sufficiently large number of learning iterations under certain conditions. Specifically, when initiating the optimization of the second term from any starting policy derived by the first term of Eq. (3), we maintain a consistent boundary between the optimal policy π^* and the learned policy π_k [61], $\|Q^{\pi_k} - Q^*\|_\infty \leq \frac{1-\gamma^k}{1-\gamma}\sqrt{2AC\epsilon} + \gamma^k \frac{R_{\max}}{1-\gamma}$. $R_{\max}$ represent the maximum value of reward function. A is the size of possible actions within the action space. The bound between performance of the optimal policy π^* and the learned policy π_k is, $\|\mathcal{L}(\pi_k) - \mathcal{L}(\pi^*)\|_\infty \leq \frac{2}{1-\gamma}\left(\frac{1-\gamma^k}{1-\gamma}\sqrt{2AC\epsilon} + \gamma^k \frac{R_{\max}}{1-\gamma}\right)$. Therefore, after executing a sufficient learning process, TrajDeleter effectively fine-tunes the policy to achieve convergence.

VI. Experimental Setup

A. Investigated Tasks and Datasets

We conduct experiments on three widely robotic control tasks, `Hopper`, `Half-Cheetah`, and `Walker2D` from MuJoCo [6], which are all commonly used in previous studies [13]. In the `Hopper` task, the objective for the agent is to maneuver a one-legged robot to move forward at the highest possible speed. Moreover, in the `Half-Cheetah` and `Walker2D` tasks, the agent is tasked with controlling a cheetah and a bipedal robot respectively, to walk forward as fast as possible. We utilized offline datasets from D4RL, a benchmark designed for offline RL algorithm evaluation. D4RL offers diverse datasets for these tasks, including *medium*, *random*, *medium-replay*, and *medium-expert*, each collected using different policies. In our experiments, we chose the *medium-expert*, an agent trained to achieve the highest performance compared to others using different datasets. Further details on tasks and chosen datasets are available in Appendix A.

B. Baselines

Retraining from Scratch (Reference). This baseline involves retraining the agent from scratch. Retraining the agent is applicable when the original training data is accessible and ensures complete trajectory removal. Generally, this method offers a precise guarantee for unlearning specific trajectories, but it is resource-intensive. This method acts as a reference for evaluating other unlearning methods.

Fine-tuning. This baseline extends the training of an agent using the dataset from which the targeted trajectories have been removed. We implement this fine-tuning process to adjust the agent's parameters with a limited number of iterations.

Random-reward. In RL, an agent's training is fundamentally guided by a reward-based paradigm [57]. Intuitively, for a specific state, if an agent receives a high reward for an action, it is more likely to choose that action again under similar states in preference to actions associated with lower rewards. This baseline edits the reward in the trajectories selected for unlearning by assigning them random rewards. Then, the original agent is fine-tuned on this modified dataset for the unlearning process. In our experiments, random rewards are generated by sampling from a uniform distribution, where the maximum and minimum values are the highest and lowest rewards observed in the entire offline dataset.

C. Implementation and Experiment Platforms

TrajDeleter is designed to be agent-agnostic: it should handle unlearning requests for agents trained using various offline RL algorithms. We select six offline RL algorithms that are prevalently used in offline RL community [52]. Specifically, we select bootstrapping error accumulation reduction (BEAR) [37], batch-constrained deep Q-learning (BCQ) [19], conservative Q-learning (CQL) [38], implicit Q-learning (IQL) [36], policy in the latent action apace with perturbation (PLAS-P) [76], and twin delayed deep deterministic policy gradient plus behavioral cloning (TD3PlusBC) [18]. We elaborate on the details of investigated algorithms in Appendix C. We use the open-source repository for implementation [52]. For more details on implementation and experiment platforms, please refer to Appendix D.

D. Evaluation Metrics

We introduce metrics used to assess the TrajDeleter from the perspectives mentioned in Appendix B.

Precision, Recall and F-1 scores. These metrics are used to assess the effectiveness of trajectory removal, i.e., the *efficacy* of unlearning [10], [66]. We define "true positives" (TP) as the trajectories that are actually included in the training dataset of the agent, and "false negatives" (FN) refer to the trajectories that are incorrectly marked as part of the training dataset, as identified by TrajAuditor. False negatives (FN) denote those trajectories that, despite being part of the training dataset, are erroneously classified as not included, according to the evaluation by TrajAuditor.

TABLE I: Precision, recall, and F1-score of TrajAuditor. We evaluate the performance at different unlearning rates using exact unlearning methods, i.e., retraining from scratch (reference).

Tasks	Algorithms	Unlearning rates											
		0.01			0.05			0.1			0.15		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Hopper	BEAR	100.0 %	85.43 %	0.92	99.68 %	84.37 %	0.91	100.0 %	85.15 %	0.92	99.75 %	84.77 %	0.92
	BCQ	100.0 %	81.79 %	0.90	100.0 %	80.14 %	0.89	99.77 %	80.75 %	0.89	98.06 %	80.01 %	0.88
	CQL	100.0 %	84.70 %	0.92	100.0 %	84.92 %	0.92	100.0 %	84.46 %	0.92	100.0 %	85.57 %	0.92
	IQL	99.75 %	83.25 %	0.91	98.98 %	84.06 %	0.91	94.18 %	84.54 %	0.91	95.79 %	84.77 %	0.90
	PLAS-P	100.0 %	54.01 %	0.71	100.0 %	71.25 %	0.85	98.69 %	78.46 %	0.87	88.24 %	81.45 %	0.83
	TD3PlusBC	99.03 %	85.84 %	0.92	98.30 %	83.28 %	0.90	96.00 %	83.93 %	0.90	93.33 %	84.69 %	0.88
	Average	**99.80** %	**79.17** %	**0.88**	**99.49** %	**81.34** %	**0.90**	**97.77** %	**82.21** %	**0.90**	**95.86** %	**83.54**%	**0.89**
Half-Cheetah	BEAR	100.0 %	83.18 %	0.91	99.68 %	84.37 %	0.91	100.0 %	84.31 %	0.91	100.0 %	82.50 %	0.91
	BCQ	100.0 %	80.54 %	0.88	100.0 %	75.89 %	0.86	99.75 %	76.89 %	0.87	99.25 %	80.41 %	0.89
	CQL	100.0 %	83.00 %	0.91	100.0 %	77.50 %	0.87	100.0 %	80.00 %	0.88	99.72 %	81.42 %	0.89
	IQL	100.0 %	77.65 %	0.87	100.0 %	79.45 %	0.88	100.0 %	81.72 %	0.90	100.0 %	80.72 %	0.89
	PLAS-P	100.0 %	62.55 %	0.76	100.0 %	72.58 %	0.84	100.0 %	65.47 %	0.78	100.0 %	62.04 %	0.76
	TD3PlusBC	99.45 %	81.13 %	0.89	100.0 %	80.21 %	0.89	100.0 %	80.08 %	0.89	100.0 %	81.50 %	0.90
	Average	**99.91** %	**77.84** %	**0.87**	**99.95** %	**78.33** %	**0.88**	**99.83** %	**79.76** %	**0.88**	**98.31** %	**78.73** %	**0.88**
Walker2D	BEAR	99.73 %	82.95 %	0.91	99.61 %	81.32 %	0.90	100.0 %	83.54 %	0.91	99.19 %	83.33 %	0.91
	BCQ	100.0 %	78.25 %	0.88	99.84 %	76.85 %	0.87	99.32 %	78.36 %	0.86	98.88 %	76.85 %	0.86
	CQL	100.0 %	81.99 %	0.89	100.0 %	85.45 %	0.91	99.80 %	82.55 %	0.90	99.35 %	82.27 %	0.90
	IQL	99.75 %	83.25 %	0.91	98.15 %	84.42 %	0.90	97.32 %	74.75 %	0.85	99.04 %	73.71 %	0.84
	PLAS-P	99.77 %	80.18 %	0.89	99.67 %	78.74 %	0.88	100.0 %	79.95 %	0.88	99.28 %	76.89 %	0.87
	TD3PlusBC	99.76 %	84.68 %	0.92	99.15 %	83.78 %	0.90	99.65 %	83.77 %	0.91	99.26 %	85.26 %	0.92
	Average	**99.84** %	**81.88** %	**0.90**	**99.57** %	**81.76** %	**0.89**	**99.00** %	**79.72** %	**0.88**	**98.51** %	**79.80** %	**0.87**

Precision is the number of TP over the number of TP plus the number of FP, i.e., $\frac{\text{TP}}{\text{TP+FP}} \times 100\%$. Besides, recall is defined as the number of TP over the number of TP plus the number of FN, i.e., $\frac{\text{TP}}{\text{TP+FN}} \times 100\%$. The F1 score, defined as $\frac{2\cdot\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}} \times 100\%$, represents the harmonic mean of precision and recall. A higher F1 score denotes a more proficient method for testing trajectory removal.

Averaged Cumulative Return. An unlearning method is useful only if it maintains performance levels comparable to the original agent. Hence, we consider the *fidelity* to be the second performance measure, apply *Averaged Cumulative Return* to the quantity of the agent's performance. An agent interacts with the environment, producing a test trajectory denoted by τ. The cumulative return of this trajectory is defined as $R(\tau) = \sum_{i=0}^{|\tau|} r_i$. We collect a set of test trajectories $\mathcal{T}$. The agent's performance is then quantified by the average of the cumulative returns, i.e., $\frac{1}{|\mathcal{T}|} \sum_{\tau \in \mathcal{T}} R(\tau)$. Consistent with the evaluation in previous works [17], [52], our experiments calculate the average cumulative return over 100 test trajectories. A higher return signifies superior agent performance.

VII. Empirical Evaluations

This section evaluates the effectiveness of TrajDeleter by answering the following three research questions (RQs),

- **RQ1.** Does the TrajAuditor effectively verify the efficacy of unlearning?
- **RQ2.** How effective is TrajDeleter?
- **RQ3.** How do hyper-parameters affect the performance of TrajDeleter?

RQ1 initially determines whether TrajAuditor for offline reinforcement unlearning can identify the presence of a trajectory in the training dataset. Then, in the RQ2, we apply TrajDeleter along with baselines to unlearn specific trajectories across various tasks and offline RL algorithms, to present the effectiveness of TrajDeleter from four perspectives described in Section B. In the RQ3, we investigate the influence of hyper-parameters on the performance of TrajDeleter.

RQ1. Does the proposed TrajAuditor *effectively verify the efficacy of unlearning?*

Experiment Design. We define the *unlearning rate* as the proportion of trajectories that need to be forgotten within the entire dataset (i.e., original dataset). We split the entire offline dataset, collected for the mixture of *medium-expert* offline dataset of each task, to the remaining dataset and unlearning dataset across different unlearning rates, i.e., $\{0.01, 0.05, 0.10, 0.15\}$. We ensure that the unlearning and the remaining datasets are collected using the same policy. The original agent is trained on the complete offline dataset, while the unlearned agent is trained from scratch using the remaining dataset with 1×10^6 timesteps. Thus, the unlearning dataset is definitely excluded from the training dataset of unlearned agents. The number of shadow agents is 5; each fine-tuned for only 5×10^3 timesteps starting from original agents. We perturb the states of the trajectories over 4 rounds using noise sampled from a Gaussian distribution with a mean of 0 and a standard deviation of 0.05.

Result Analysis. Table I presents the precision, recall, and F1-scores achieved by TrajAuditor across agents, each trained using distinct offline RL algorithms and subjected to the dataset with various unlearning rates. This table shows that TrajAuditor consistently attains high F1-scores of 0.85 across agents we tested, showing its efficacy and robustness. The average precision across the investigated tasks and unlearning rates stands at 98.9%, significantly higher than the recall rate of 80.3%. We derive these results by averaging the precision and recall values presented in the last row of Table I. These results suggest that TrajAuditor is

TABLE II: Percentages of positive predictions by TRAJAUDITOR for the unlearning dataset post unlearning method application. D_m indicates that the remaining dataset is not subjected to unlearning. The $D_{f,0.01}$ and $D_{f,0.05}$ denote the unlearning dataset with the size of 1% and 5% of the original dataset.

Tasks	Algorithms	Retraining (reference)			Fine-tuning			Random-reward			TrajDeleter		
		$\mathcal{D}_m$	$\mathcal{D}_{f,0.01}$	$\mathcal{D}_{f,0.05}$	$\mathcal{D}_m$	$\mathcal{D}_{f,0.01}$	$\mathcal{D}_{f,0.05}$	$\mathcal{D}_m$	$\mathcal{D}_{f,0.01}$	$\mathcal{D}_{f,0.05}$	$\mathcal{D}_m$	$\mathcal{D}_{f,0.01}$	$\mathcal{D}_{f,0.05}$
Hopper	BEAR	89.0 %	2.1 %	4.2 %	87.8 %	90.9 %	79.4 %	84.2 %	62.2	30.5 %	85.1 %	0.0 %	0.0 %
	BCQ	82.1 %	0.0 %	2.9 %	83.4 %	40.6 %	47.1 %	83.2 %	36.4 %	28.4 %	80.2 %	0.0 %	0.0 %
	CQL	85.2 %	0.0 %	0.0 %	84.9 %	80.6 %	75.1 %	88.1 %	77.2 %	43.5 %	83.2 %	0.0 %	0.0 %
	IQL	88.1 %	0.0 %	5.4 %	84.5 %	81.1 %	82.4 %	84.5 %	58.4 %	36.7 %	85.2 %	0.0 %	0.0 %
	PLAS-P	45.8 %	0.0 %	0.0 %	54.3 %	77.6 %	80.7 %	57.8 %	57.1 %	46.7 %	56.3 %	40.5 %	47.5 %
	TD3PlusBC	82.4 %	7.2 %	9.8 %	83.1 %	75.4 %	74.3 %	83.8 %	77.3 %	60.1 %	84.3 %	0.0 %	0.0 %
	Average	**78.8** %	**1.6** %	**3.7** %	**79.7** %	**74.4** %	**73.2** %	**80.3** %	**61.4** %	**41.8** %	**79.1** %	**6.8** %	**7.9** %
Half-Cheetah	BEAR	80.1 %	0.0 %	6.5 %	82.8 %	60.8 %	51.3 %	82.9 %	40.5 %	10.2 %	80.1 %	0.0 %	0.0 %
	BCQ	80.0 %	0.0 %	0.0 %	82.7 %	70.0 %	28.0 %	79.5 %	0.0 %	0.0 %	83.1 %	0.0 %	0.0 %
	CQL	82.7 %	0.0 %	0.0 %	81.7 %	66.6 %	52.6 %	81.3 %	24.6 %	0.0 %	77.9 %	0.0 %	0.0 %
	IQL	78.9 %	0.0 %	0.0 %	80.7 %	0.0 %	0.0 %	84.4 %	0.0 %	0.0 %	82.0 %	0.0 %	0.0 %
	PLAS-P	53.7 %	0.0 %	0.0 %	64.2 %	74.0 %	87.2 %	56.7 %	74.5 %	77.2 %	58.3 %	1.8 %	3.5 %
	TD3PlusBC	75.3 %	0.0 %	0.0 %	75.6 %	45.5 %	54.7 %	82.2 %	80.0 %	25.6 %	78.2 %	0.0 %	0.0%
	Average	**75.1** %	**0.0** %	**1.1** %	**78.1** %	**52.8** %	**45.6** %	**77.8** %	**36.6** %	**18.8** %	**76.6** %	**0.3** %	**0.6** %
Walker2D	BEAR	82.5 %	6.2 %	5.2 %	83.6 %	69.5 %	71.9 %	84.5 %	29.5 %	8.6 %	84.3 %	0.0 %	0.6 %
	BCQ	80.2 %	0.0 %	1.4 %	78.5 %	40.6 %	31.5 %	79.8 %	21.6 %	0.0 %	79.5 %	0.0 %	0.2 %
	CQL	79.3 %	0.0 %	0.9 %	84.2 %	52.7 %	64.5 %	81.5 %	46.5 %	5.3 %	82.3 %	0.0 %	0.0 %
	IQL	78.9 %	0.0 %	0.0 %	78.5 %	65.2 %	49.9 %	78.0 %	48.6 %	3.5 %	77.5 %	0.0 %	0.3 %
	PLAS-P	80.6 %	1.4 %	4.3 %	81.6 %	66.7 %	70.8 %	81.9 %	69.6 %	74.5 %	83.2 %	51.6 %	60.7 %
	TD3PlusBC	84.6 %	1.5 %	2.1 %	84.3 %	79.7 %	58.1 %	82.8 %	79.5 %	10.5 %	84.7 %	0.0 %	1.3 %
	Average	**81.0** %	**1.5** %	**2.3** %	**81.8** %	**62.4** %	**57.8** %	**81.4** %	**49.2** %	**17.1** %	**81.9** %	**8.6** %	**10.3** %

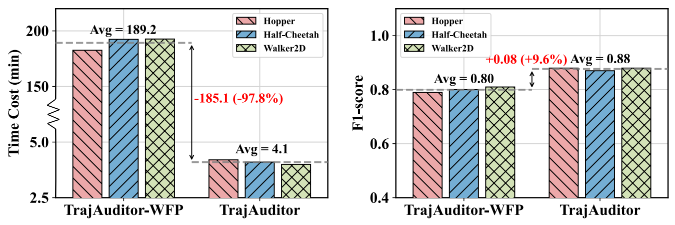

Fig. 3: The time costs (left) and F1-scores (right) achieved by (1) our proposed TRAJAUDITOR, (2) TRAJAUDITOR without the fine-tuning component and excluding state perturbations. The 'Avg' means the average of three tasks.

more likely to identify tested trajectories not included in the agents' training dataset. Offline RL algorithms often suffer from overestimating value function training [19], [38], [76], leading to inaccurate computations of the value vectors for trajectories, mistakenly categorizing them as excluded from the training dataset. We study the robustness of TRAJAUDITOR in Appendix E1.

We conduct ablation studies to highlight the importance of generating shadow agents by fine-tuning original agents and applying state perturbation in TRAJAUDITOR. TRAJAUDITOR-WFP denotes the removal of state perturbations in the tested trajectory and fine-tuning of agents from the original agents. By directly training shadow models from scratch without fine-tuning or state perturbation, TRAJAUDITOR is essentially reduced to ORL-AUDITOR. Thus, we equate TRAJAUDITOR-WFP with ORL-AUDITOR. Figure 3 shows that, on average across three tasks, TRAJAUDITOR reduces time costs by 97.8% while achieving an F1-score that is 0.08 higher compared to TRAJAUDITOR-WFP. These results present that our method can enhance the TRAJAUDITOR's performance.

Answers to RQ1: TRAJAUDITOR achieves average F1-scores of 0.88, 0.87, and 0.88 across three tasks, tested at four different unlearning rates. These results present that TRAJAUDITOR accurately identifies trajectories involved in agents' training datasets. It is a simple yet efficient tool for assessing the efficacy of offline reinforcement unlearning.

RQ2. How effective is TRAJDELETER*?*

Experiment Design. We evaluate TRAJDELETER from four perspectives as we described in Section B. Given that the unlearning dataset is typically a small portion of the original dataset, we examine scenarios where 1% and 5% of the offline dataset (i.e., unlearning rates of 0.01 and 0.05) are segregated as unlearning datasets. The remaining portion of the dataset is then considered as the remaining dataset. For retraining from scratch, we follow the methodology detailed in RQ1. The retrained agents serve as a reference for comparing other unlearning methods (i.e., fine-tuning, random-reward, and TRAJDELETER). The unlearning steps for the other methods are set at 1×10^4, amounting to only 1% of the steps required for retraining (1×10^6). The durations for "forgetting" (K) and "convergence training" (H) are set at 8000 and 2000 timesteps, respectively. Additionally, we set the balancing factor λ at 1 (as detailed in Section V-C).

Result Analysis. We evaluate the performance of TRAJDELETER from four distinct perspectives as follows.

Efficacy evaluation. Table II presents the Percentage of Positive pRedictions (PPR) made by TRAJAUDITOR, reflecting the extent to which the target dataset continues to influence the agents after the implementation of various unlearning methods. The last two columns in Table II present that excluding the average values, only 40.5%, 47.5%, 51.6%, and 60.7% of the settings exceeded the 3.5% in these 36 values. These results show that TRAJDELETER can efficiently eliminate the impact of target trajectories on agents, with 32 out of 36 settings exhibiting a PPR below 3.5% after unlearning.

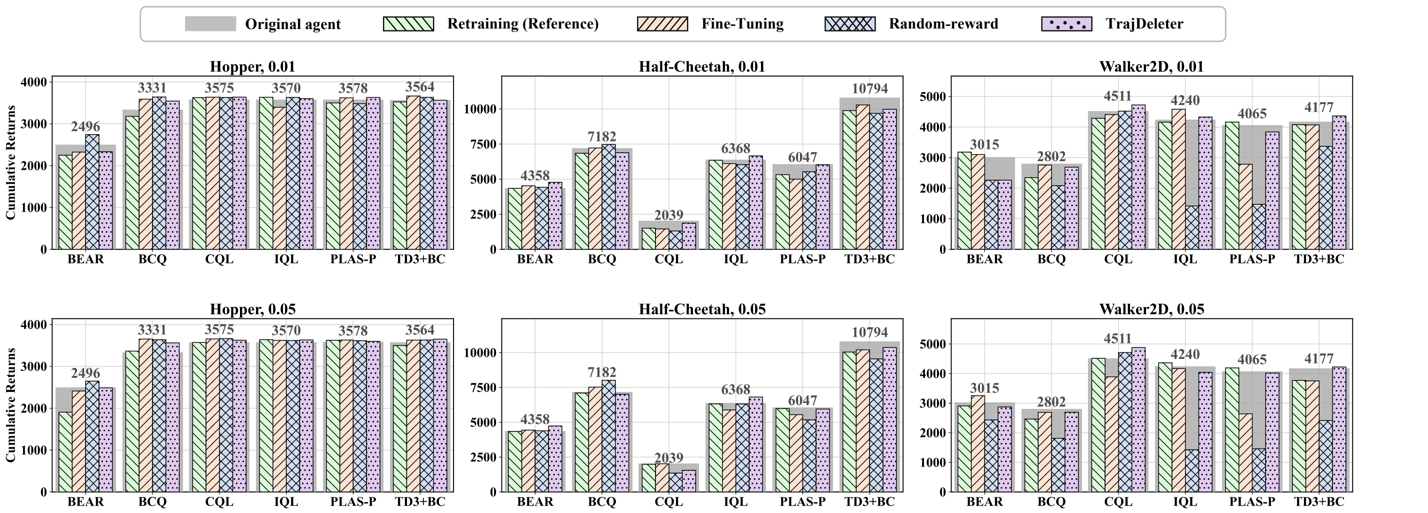

Fig. 4: The cumulative returns are averaged over 100 test trajectories, collected using unlearned agents trained with 5 different random seeds. "0.01" and "0.05" represent the unlearning rates. The values beyond the gray bars indicate the cumulative returns of the original agents.

TABLE III: The time costs required for unlearning across TRAJDELETER and its baselines in the three tasks.

Methods	Tasks		
	`Hopper`	`Half-Cheetah`	`Walker2D`
Retraining	200.4 min	223.4 min	199.4 min
Fine-tuning	3.7 min	3.6 min	3.2 min
Random-reward	3.5 min	3.4 min	3.2 min
TrajDeleter	3.4 min	3.2 min	3.4 min

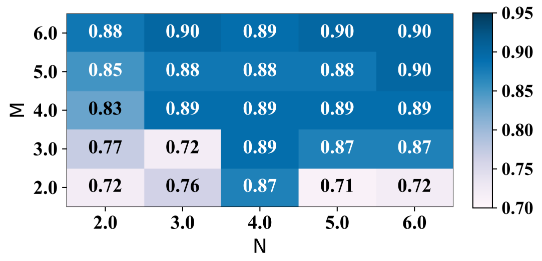

Fig. 5: The average F1-score of TRAJAUDITOR (the unlearning rate is 0.01) for the `Half-Cheetah` task across M and N.

Table II shows that the variance in PPR made by agents unlearned using TRAJDELETER is minimal compared to the outcomes achieved by retraining from scratch. Specifically, after retraining, TRAJAUDITOR predicts that on average, only 2.7%, 0.55%, and 1.9%[2] of the unlearned dataset continues to influence the agent for the three tasks under investigation. In contrast, for TRAJDELETER, the average PPRs after unlearning are 7.35%, 0.45%, and 9.45%. These results are significantly lower than those of the baseline methods – after fine-tuning, the PPRs are 73.8%, 44.2%, and 60.1%; while following the random-reward approach, the PPRs stand at 51.6%, 27.7%, and 33.2%. By calculating the difference in average PPRs across the remaining dataset $\mathcal{D}_m$ for three tasks, the PPRs for the remaining dataset exhibit only slight variations post-retraining, with a mere 0.9% increase compared to the retraining method. TRAJDELETER maintains the integrity of the trajectories' memory within the remaining dataset.

Fidelity evaluation. Figure 4 displays the averaged cumulative returns, calculated by averaging both the mean and variance across 100 test trajectories. As illustrated in Figure 4, the unlearned agents have a comparable performance level compared to those of the agents retrained from scratch. Specifically, the average cumulative returns demonstrate a marginal difference of 2.2%, 0.9%, and 1.6% between the unlearned agents using TRAJDELETER and those subjected to the retraining method. These results suggest that TRAJDELETER does not negatively impact the performance of unlearned agents in real-world interactions, demonstrating the high practicality of our method. More analysis are provided in Appendix D2.

Efficiency evaluation. Table III presents the averaged time costs required for unlearning across TRAJDELETER and its baselines in three tasks. TRAJDELETER requires only 3.4, 3.2, and 3.4 minutes for tasks, significantly less than the 200.4, 223.4, and 100.4 minutes required for retraining from scratch. These results present that TRAJDELETER requires only 1.5% of the time compared to retraining from scratch.

Agent agnostic evaluation. We have selected six offline RL algorithms to train agents, determining the efficacy of TRAJDELETER in unlearning specific trajectories. Most agents (32 out of 36) can effectively forget the target trajectories without substantial degradation in performance. However, Table II shows that TRAJDELETER achieves over 40.5% PPR when unlearning agents trained using the PLAS-P algorithm in `Hopper` and `Walker2D`. These results could be attributed to inaccuracies in TRAJAUDITOR. As shown in Table I, TRAJAUDITOR gets explicitly low recall rates (e.g., with an unlearning rate of 0.01, the recall rate for the `Hopper` task is 54.01%) for agents trained using PLAS-P.

[2]Note that these figures are calculated as the mean value of PPR across two unlearning rates: (1.6% + 3.7%) / 2 = 2.7%, (0.0% + 1.1%) / 2 = 0.55%, and (1.5% + 2.3%) / 2 = 1.9%. The other percentages mentioned in this paragraph are derived using the same method.

TABLE IV: The relative changes in percentages of positive predictions of TrajAuditor on remaining dataset $\mathcal{D}_m$ and unlearning dataset $\mathcal{D}_f$, and average returns of unlearned agents just using "forgetting", when compared with its performance after doing "convergence training". The symbols '↓', '↑', and '-' denote decreases, increases, and no changes.

Tasks	Algorithms	D_m	Unlearning rates			
			0.01		0.05	
			D_f	Returns	D_f	Returns
Hopper	BEAR	↓2.1%	- 0.0 %	↓1499	- 0.0 %	↓1429
	BCQ	↓7.2%	- 0.0 %	↓2271	- 0.0 %	↓2562
	CQL	↓1.3%	- 0.0 %	↓1903	- 0.0 %	↓852
	IQL	- 0.0%	- 0.0 %	↓345	- 0.0 %	↓980
	PLAS-P	↓45.5%	↑0.7 %	↓1783	- 0.0 %	↓1732
	TD3PlusBC	↓1.4%	- 0.0 %	↑20	- 0.0 %	↓75
	Average	↓9.6%	↑ 0.1%	↓1297	-0.0%	↓1272
Half-Cheetah	BEAR	- 0.0%	- 0.0 %	↑772	- 0.0 %	↑611
	BCQ	↓13.2%	- 0.0 %	↑535	- 0.0 %	↑451
	CQL	↓4.2%	- 0.0 %	↓100	- 0.0 %	↓344
	IQL	- 0.0%	- 0.0 %	↑382	- 0.0 %	↑431
	PLAS-P	↓1.9%	↑0.1 %	↓102	- 0.0 %	↓105
	TD3PlusBC	↓32.1%	- 0.0 %	↑218	- 0.0 %	↓312
	Average	↓8.6%	- 0.0%	↑284	- 0.0%	↑122
Walker2D	BEAR	↓5.9%	- 0.0 %	↓134	- 0.0 %	↓755
	BCQ	↓6.1%	- 0.0 %	↑258	- 0.0 %	↑104
	CQL	↓10.1%	- 0.0 %	↓2712	- 0.0 %	↓2064
	IQL	- 0.0 %	- 0.0 %	↓101	- 0.0 %	↑289
	PLAS-P	↓2.1%	↑3.1 %	↑121	↑2.9 %	↓885
	TD3PlusBC	- 0.0%	- 0.0 %	↓488	- 0.0 %	↓144
	Average	↓4.0%	↑0.5%	↓509	↑0.5%	↓576

Answers to RQ2: The average PPRs after unlearning are 7.35%, 0.45%, and 9.45% for investigated tasks, meaning that TrajDeleter unlearns 92.7%, 99.5%, 90.5% of the target trajectories. Yet TrajDeleter maintains robust performance in actual environment interactions. The fact that 32 out of 36 settings display a PPR below 3.5% shows the superior agent-agnostic capability of TrajDeleter.

RQ3. How do hyper-parameters affect the TrajDeleter*?*

Experiment Design. This section first explores the impact of the forgetting learning steps, K, and the balancing factor, λ, on the unlearning performance of TrajDeleter. The forgetting steps, $K = \{0, 2000, 4000, 6000, 8000\}$. The balancing factor λ is to balance the unlearning on forgetting datasets and training on remaining datasets, thereby preventing deterioration in the agent's performance. We have configured it with values $\{0.25, 0.5, 0.75, 1.0, 1.5\}$. Then, to explore the impact of the number of shadow models and perturbation rounds in TrajAuditor, denoted as N and M, we vary these parameters across the values $\{2, 3, 4, 5, 6\}$.

Result Analysis. The first, second, and third subfigures in Figure 6 present the average trends in cumulative reward and Percentage of Positive pRedictions (PPR) as obtained by TrajAuditor for the unlearned agents across varying forgetting training steps. As the number of forgetting steps increases from 0 to 8000, the average PPRs of unlearned agents decrease from 74.4%, 52.8%, and 62.4% to 7.9%, 0.6%, and 10.5% for the three tasks, respectively. In contrast, the agents' performance remains consistent. As illustrated in the fourth subfigure of Figure 6, when the forgetting training is adequate, the λ exerts minimal influence on the performance

TABLE V: The average returns of the poisoned and unlearned agents. The values in parentheses are the relative changes compared to agents retrained on a non-poisoned dataset.

Methods	Poisoning rates	Tasks		
		Hopper	Half-Cheetah	Walker2D
Retraining	-	3357	5761	3868
Poisoning	0.01	2169 (**-35.4%**)	5263 (**-8.6%**)	3127 (**-19.2%**)
	0.05	2059 (**-38.6%**)	5455 (**-5.3%**)	3158 (**-18.4%**)
	Average	2114 (**-37.0%**)	5359 (**-7.0%**)	3158 (**-18.8%**)
Trajdeleter	0.01	3299 (**-1.7%**)	6029 (**+4.6%**)	3949 (**+2.1%**)
	0.05	3421 (**+1.9%**)	5679 (**-1.4%**)	3855 (**-0.3%**)
	Average	3360 (**+0.1%**)	5854 (**+1.6%**)	3902 (**+0.9%**)

of TrajDeleter. Moreover, the last subfigure in Figure 6 indicates that when the number of forgetting steps is small, an increase in λ can enhance the unlearning efficiency of TrajDeleter. Overall, K exhibits greater sensitivity than λ, suggesting that the primary focus should be tuning K. Figure 5 presents the average F1-score of TrajAuditor across agents trained with six offline RL algorithms for the Half-Cheetah task, varying the parameters M and N. In this figure, we observe that as the numbers of M and N increase, the F1 scores rise from 0.71 to a plateau of 0.90. We provide a more comprehensive analysis in Appendix D3.

Answers to RQ3: With an increase in the number of forgetting steps, the performance of TrajDeleter improves, leading to the unlearned agent forgetting more trajectories. When the number of forgetting steps is high, TrajDeleter exhibits minimal sensitivity to changes in the balancing factor's value.

VIII. Discussions

A. Ablation Studies

This section conducts ablation studies to emphasize the importance of "convergence training" in the effectiveness of TrajDeleter. Our experiments focus solely on implementing "forgetting" training with TrajDeleter, aimed at unlearning the 1% dataset, without engaging in convergence training. In this experiment, the unlearning rates are set at 0.01 and 0.05. The results are illustrated in Table IV. We observe that **"convergence training" leads to stronger TrajDeleter**. We attribute the observed outcomes to the following reasons: In the absence of convergence training for the three tasks: (1) at an unlearning rate of 0.05, the average cumulative returns of agents show changes of 1272, 122, and 576, respectively, with corresponding PPR changes on the unlearning dataset $\mathcal{D}_f$ of 0.0%, 0.0%, and 0.5%; (2) TrajAuditor experiences reductions in the average percentages of positive predictions by 9.6%, 8.6%, and 4.0% on the remaining dataset. These findings suggest that while convergence training has a minimal impact on the unlearning trajectories, it assists in enhancing the performance of the unlearned agent. Furthermore, it helps in preventing adverse effects on the trajectories within the remaining dataset.

B. Defending Against Trajectory Poisoning

We also conduct experiments to investigate the effectiveness of TrajDeleter in defending against trajectory poisoning attacks. We poison the original dataset by adjusting the action

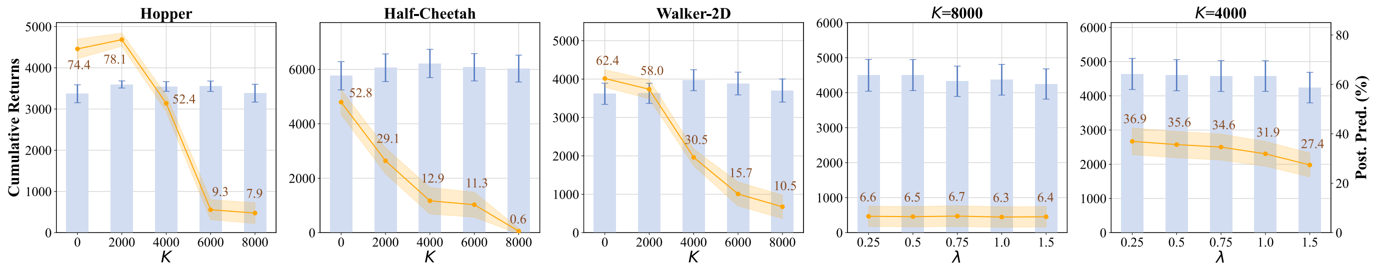

Fig. 6: The influence of forgetting learning steps, K, and the balancing factor, λ, on the unlearning performance of TrajDeleter. Yellow points denote the average percentages of Positive Predictions (Post. Pred.) by TrajAuditor for unlearned agents, aligning with the *left* y-axis. Shaded areas denote the standard deviation. Light blue bars illustrate the average returns, corresponding to the *right* y-axis, with the error bars meaning standard deviations. The unlearning rate is 0.05.

values in the trajectories to be 1.5 times their mean value. This modification is applied to only 5% of the whole dataset. *Poisoning rates* denote the fraction of generated poisoned trajectories in the whole dataset.

All other experimental settings remain consistent with those used for training the original agent. Table V shows the averaged returns of the poisoned agents, compared with those of agents after unlearning poisoned trajectories. After training the agents on a poisoned dataset, we observe a decrease in their average performance, which is 37.0%, 7.0%, and 18.8% across the three tasks under investigation. TrajDeleter can mitigate the effects of poisoning in agents, thereby enhancing their performance to match that of agents who have been retrained on a non-poisoned dataset from scratch.

C. The Impact of Repeated Unlearning

This section explores how repeated learning and unlearning of the same trajectories impact the averaged returns of agents. We repeatedly unlearn and learn the same trajectories with unlearning rates of 0.01 and 0.05, doing so $\{1, 3, 7, 10\}$ times respectively, and record the averaged returns of agents trained using six offline RL algorithms in Table VI. Referring to the experiments in RQ1 and RQ2, the learning and unlearning phases consist of 1×10^6 and 1×10^4 timesteps, respectively. Table VI shows no decreasing trend in the agent's cumulative returns with an increasing number of repeated learning cycles. In particular, compared to the original agents, the largest relative change in this table for agents that have repeatedly learned and removed the same trajectories is 4.0%. In most cases, the relative changes are less than 2%. While the agent's performance exhibits fluctuations, this variability is a natural aspect of training, commonly observed in offline RL [40]. We believe that repeatedly unlearning and relearning the same trajectories do not lead to damage to agents' performance.

D. Limitation

Legal Uncertainty. The approach most aligned with requirements of data protection regulations is to retrain the ML model. However, this approach is overly expensive in modern ML. Approximate unlearning has been proposed as a compromise that does not fully delete every impact of the data to be deleted, but is much more efficient [64], [74], and TrajDeleter follows these works. Certified unlearning [64] offers a theoretical

TABLE VI: The averaged returns of the agents which are measured through the repeatedly addition and removal of learning specific trajectories. The values in parentheses represent the relative changes compared to the original agent's return.

Times	Unlearning rates	Tasks		
		Hopper	Half-Cheetah	Walker2D
1	0.01	3473 (**+3.4%**)	5788 (**+0.4%**)	3946 (**+2.5%**)
	0.05	3493 (**+4.0%**)	5692 (**-1.1%**)	3929 (**+1.6%**)
3	0.01	3294 (**-1.8%**)	5832 (**+1.2%**)	3829 (**-1.0%**)
	0.05	3401 (**+1.3%**)	5701 (**-1.0%**)	3782 (**-2.2%**)
7	0.01	3449 (**+2.7%**)	5678 (**-1.4%**)	3892 (**+0.6%**)
	0.05	3378 (**+0.6%**)	5636 (**-2.1%**)	3832 (**-0.9%**)
10	0.01	3411 (**+1.6%**)	5712 (**-0.8%**)	3910 (**+1.1%**)
	0.05	3409 (**+1.5%**)	5865 (**+1.8%**)	3783 (**-2.2%**)

guarantee that the output of unlearned models closely matches that of retrained models, potentially aiding legal compliance. We discuss more in Appendix E4.

Potential Attacks. One potential attack is recovery, where an adversary attempts to reconstruct individual data entries from the training datasets of models [3]. While unlearning complicates the data recovery process, the frequent application of unlearning and subsequent parameter modifications increase the difficulty for attackers attempting to retrieve specific data. However, preventing data recovery from unlearned models is a challenging problem [9], [43], and this attack is possible even for retraining [3]. In some cases, model owners may deceive users by not performing unlearning. Even when model owners genuinely implement unlearning, previous research has suggested that the output distributions of both learned and unlearned models can be utilized to construct feature vectors to train the attack model [43].

Effectiveness Uncertainty. It is noticed that as an approximate method, the unlearning effect of TrajDeleter relies on the effectiveness of the trajectories auditing method, i.e., TrajAuditor; this is a common practice [64], [66]. We advocate enhancing the auditor tool to improve the reliability of unlearning effectiveness. We discuss more in Appendix E3.

IX. Related Works

We discuss related work briefly here, and we provide a more comprehensive discussion in Appendix F.

Offline RL for Real Applications Recently, offline RL systems work brilliantly on a wide range of real-world fields, including healthcare [46], [15], energy management systems [71], [75], and autonomous driving [72], [28]. In healthcare, online RL is not suitable, as it is ethically and practically problematic to experiment with patients' health. Mila et al. [46] used advanced offline RL to develop a policy for sepsis treatment optimization. In various areas, offline RL methods are more efficient than online RL methods [71], [35].

Deep Machine Unlearning Deep machine unlearning [5], [7], [64], [59] refers to eliminating the knowledge of specific data point(s) on the already trained Deep Neural Networks (DNNs). In general, deep machine unlearning is categorized into two main groups: exact unlearning [5], [29], [67] and approximate unlearning methods [23], [22], [11]. Exact unlearning involves retraining the DNN from scratch without the data meant to be forgotten, which is computationally demanding due to large datasets [53]. Bourtoule et al. [5] proposed the SISA method by splitting the dataset into non-overlapping shards, allowing retraining on just one shard. Unlike exact unlearning, approximate unlearning estimates DNN parameters similar to retraining from scratch [66], [23], [21], [59]. Various recent works also studied certified unlearning definitions [32], [34].

Certified Unlearning Certified unlearning ensures that unlearned models are theoretically indistinguishable from those retrained from scratch [32], [34], [64]. Guo et al.[32] have made the distributions of unlearned and retrained models nearly identical. Warnecke et al. [64] introduced a certified unlearning for the features and labels unlearning.

X. Conclusions

This paper introduces TrajDeleter, the first practical trajectory-level unlearning method designed specifically for offline RL agents. TrajDeleter enables agents to erase the influence of the target trajectory and "forget" it. This paper emphasizes approximate unlearning, which focuses more on classic supervised learning. To verify if trajectories are truly forgotten, we introduce TrajAuditor to evaluate the success of TrajDeleter in completely removing the influence of specific trajectories from the offline RL agent, paving the path for unlearning study. For unlearning target trajectories, TrajDeleter is to prompt the agent to exhibit declining performance when encountering states linked to unlearning trajectories while preserving its original performance level for other remaining trajectories. Our evaluations present that TrajDeleter consistently forgets the trajectories efficiently while maintaining strong performance in real interactions.

Acknowledgment

We thank anonymous reviewers for valuable comments. This paper was partially supported by NSF CNS-2350333.

References

[1] N. AlHinai, "Introduction to biomedical signal processing and artificial intelligence," in *Biomedical signal processing and artificial intelligence in healthcare*. Elsevier, 2020, pp. 1–28.

[2] M. Andrychowicz, A. Raichuk, P. Stańczyk *et al.*, "What matters for on-policy deep actor-critic methods? a large-scale study," in *The International Conference on Learning Representations (ICLR)*, 2021.

[3] M. Bertran, S. Tang, M. Kearns, J. Morgenstern, A. Roth, and Z. S. Wu, "Reconstruction attacks on machine unlearning: Simple models are vulnerable," *arXiv preprint arXiv:2405.20272*, 2024.

[4] F. Boenisch, "A systematic review on model watermarking for neural networks," *Frontiers in big Data*, vol. 4, p. 729663, 2021.

[5] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo *et al.*, "Machine unlearning," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 141–159.

[6] G. Brockman, V. Cheung, L. Pettersson *et al.*, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

[7] J. Brophy and D. Lowd, "Machine unlearning for random forests," in *Proceedings of the 38th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139, 18–24 Jul 2021, pp. 1092–1104.

[8] N. Carlini, S. Chien, M. Nasr *et al.*, "Membership inference attacks from first principles," in *2022 IEEE Symposium on Security and Privacy (SP)*, 2022, pp. 1897–1914.

[9] M. Chen, Z. Zhang, T. Wang, M. Backes, M. Humbert, and Y. Zhang, "When machine unlearning jeopardizes privacy," in *Proceedings of the 2021 ACM SIGSAC conference on computer and communications security*, 2021, pp. 896–911.

[10] M. Chen, Z. Zhang, T. Wang *et al.*, "Graph unlearning," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, p. 499–513.

[11] V. S. Chundawat, A. K. Tarun, M. Mandal, and M. Kankanhalli, "Zero-shot machine unlearning," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2345–2354, 2023.

[12] J. Degrave, F. Felici, J. Buchli *et al.*, "Magnetic control of tokamak plasmas through deep reinforcement learning," *Nature*, vol. 602, no. 7897, pp. 414–419, 2022.

[13] L. Du, M. Chen, M. Sun *et al.*, "Orl-auditor: Dataset auditing in offline deep reinforcement learning," *arXiv preprint arXiv:2309.03081*, 2023.

[14] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC*, 2006, pp. 265–284.

[15] H. Emerson, M. Guy, and R. McConville, "Offline reinforcement learning for safer blood glucose control in people with type 1 diabetes," *J. Biomed. Informatics*, 2023.

[16] H. Foley, L. Fowl, T. Goldstein, and G. Taylor, "Execute order 66: Targeted data poisoning for reinforcement learning," *CoRR*, 2022. [Online]. Available: https://arxiv.org/abs/2201.00762

[17] J. Fu, A. Kumar, O. Nachum, G. Tucker, and S. Levine, "D4rl: Datasets for deep data-driven reinforcement learning," 2021.

[18] S. Fujimoto and S. S. Gu, "A minimalist approach to offline reinforcement learning," in *Advances in Neural Information Processing Systems*, vol. 34. Curran Associates, Inc., 2021, pp. 20 132–20 145.

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

[20] D. Georgiou and C. Lambrinoudakis, "Compatibility of a security policy for a cloud-based healthcare system with the EU general data protection regulation (GDPR)," *Inf.*, vol. 11, no. 12, p. 586, 2020.

[21] A. Golatkar and A. Achille, "Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations," in *ECCV 2020*, vol. 12374, 2020, pp. 383–398.

[22] A. Golatkar, A. Achille, A. Ravichandran *et al.*, "Mixed-privacy forgetting in deep networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2021, pp. 792–801.

[23] A. Golatkar, A. Achille, and S. Soatto, "Eternal sunshine of the spotless net: Selective forgetting in deep networks," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 9301–9309.

[24] M. Gomrokchi, S. Amin, H. Aboutalebi, A. Wong, and D. Precup, "Membership inference attacks against temporally correlated data in deep reinforcement learning," 2022.

[25] C. Gong, K. Li, J. Yao, and T. Wang, "Trajdeleter: Enabling trajectory forgetting in offline reinforcement learning agents," *arXiv preprint arXiv:2404.12530*, 2024.

[26] C. Gong, Z. Yang, Y. Bai *et al.*, "Mind your data! hiding backdoors in offline reinforcement learning datasets," *arXiv:2210.04688*, 2022.

[27] C. Gong, Z. Yang, Y. Bai, J. Shi, A. Sinha, B. Xu, D. Lo, X. Hou, and G. Fan, "Curiosity-driven and victim-aware adversarial policies," in *Proceedings of the 38th Annual Computer Security Applications Conference*, ser. ACSAC '22, 2022, p. 186–200.

[28] D. Graves, N. M. Nguyen, K. Hassanzadeh *et al.*, "Learning robust driving policies without online exploration," in *IEEE International Conference on Robotics and Automation, ICRA*, 2021.

[29] L. Graves, V. Nagisetty, and V. Ganesh, "Amnesiac machine learning," in *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*. AAAI Press, 2021, pp. 11 516–11 524.

[30] E. Greensmith, P. L. Bartlett, and J. Baxter, "Variance reduction techniques for gradient estimates in reinforcement learning," *J. Mach. Learn. Res.*, vol. 5, pp. 1471–1530, 2004.

[31] F. E. Grubbs, "Sample criteria for testing outlying observations," *The Annals of Mathematical Statistics*, vol. 21, no. 1, pp. 27–58, 1950.

[32] C. Guo, T. Goldstein, A. Hannun, and L. Van Der Maaten, "Certified data removal from machine learning models," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 119, 2020, pp. 3832–3842.

[33] T. Guo, S. Guo, J. Zhang, W. Xu, and J. Wang, "Efficient attribute unlearning: Towards selective removal of input attributes from feature representations," *arXiv preprint arXiv:2202.13295*, 2022.

[34] Z. Izzo, M. Anne Smart, K. Chaudhuri *et al.*, "Approximate data deletion from machine learning models," in *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, vol. 130. PMLR, 2021, pp. 2008–2016.

[35] H. Kim, M. Kim, F. Berto *et al.*, "Devformer: A symmetric transformer for context-aware device placement," in *International Conference on Machine Learning, ICML*, vol. 202, pp. 16 541–16 566.

[36] I. Kostrikov, A. Nair, and S. Levine, "Offline reinforcement learning with implicit q-learning," in *International Conference on Learning Representations*, 2022.

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

[38] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," in *NeurIPS*, 2020.

[39] M. Lejeune, "California consumer privacy act - erste ansätze einer annäherung zu prinzipien der DSGVO in den USA," *Comput. und Recht*, vol. 34, no. 9, pp. 569–576, 2018.

[40] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," 2020.

[41] Y. Li, Y. Bai, Y. Jiang *et al.*, "Untargeted backdoor watermark: Towards harmless and stealthy dataset copyright protection," *Advances in Neural Information Processing Systems*, vol. 35, pp. 13 238–13 250, 2022.

[42] Z. Li, F. Nie, Q. Sun *et al.*, "Boosting offline reinforcement learning for autonomous driving with hierarchical latent skills," 2023.

[43] H. Liu, P. Xiong, T. Zhu, and P. S. Yu, "A survey on machine unlearning: Techniques and new emerged privacy risks," *arXiv preprint arXiv:2406.06186*, 2024.

[44] Y. Ma, X. Zhang, W. Sun, and J. Zhu, "Policy poisoning in batch reinforcement learning and control," in *Advances in Neural Information Processing Systems*, 2019, pp. 14 543–14 553.

[45] V. Mnih, A. P. Badia, M. Mirza *et al.*, "Asynchronous methods for deep reinforcement learning," in *Proceedings of The 33rd International Conference on Machine Learning*, 2016, pp. 1928–1937.

[46] M. Nambiar, S. Ghosh, P. Ong *et al.*, "Deep offline reinforcement learning for real-world treatment optimization applications," in *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2023.

[47] OpenAI, I. Akkaya, M. Andrychowicz, M. Chociej *et al.*, "Solving rubik's cube with a robot hand," *CoRR*, vol. abs/1910.07113, 2019.

[48] X. Pan, W. Wang, X. Zhang, B. Li, J. Yi, and D. Song, "How you act tells a lot: Privacy-leaking attack on deep reinforcement learning." in *AAMAS*, vol. 19, no. 2019, 2019, pp. 368–376.

[49] V. M. Panaretos and Y. Zemel, "Statistical aspects of wasserstein distances," *Annual review of statistics and its application*, 2019.

[50] R. F. Prudencio, M. R. O. A. Maximo, and E. L. Colombini, "A survey on offline reinforcement learning: Taxonomy, review, and open problems," *IEEE Transactions on Neural Networks and Learning Systems*, 2023.

[51] A. Sekhari, J. Acharya, G. Kamath, and A. T. Suresh, "Remember what you want to forget: Algorithms for machine unlearning," *Advances in Neural Information Processing Systems*, pp. 18 075–18 086, 2021.

[52] T. Seno and M. Imai, "d3rlpy: An offline deep reinforcement learning library," *Journal of Machine Learning Research*, vol. 23, pp. 1–20, 2022.

[53] M. Shoeybi, M. Patwary, R. Puri *et al.*, "Megatron-lm: Training multi-billion parameter language models using model parallelism," *CoRR*, vol. abs/1909.08053, 2019.

[54] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 3–18.

[55] B. Singh, R. Kumar, and V. P. Singh, "Reinforcement learning in robotic applications: a comprehensive survey," *Artif. Intell. Rev.*, 2022.

[56] P. Sodhi, F. Wu, E. R. Elenberg *et al.*, "On the effectiveness of offline RL for dialogue response generation," in *International Conference on Machine Learning, ICML*, vol. 202, 2023, pp. 32 088–32 104.

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

[58] Tessian, "How the great resignation is creating more security challenges." 2021. [Online]. Available: https://www.tessian.com/blog/how-the-great-resignation-is-creating-more-security-challenges/

[59] A. Thudi, G. Deza, V. Chandrasekaran *et al.*, "Unrolling SGD: understanding factors influencing machine unlearning," in *7th IEEE European Symposium on Security and Privacy*. IEEE, 2022, pp. 303–319.

[60] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.

[61] S. Tosatto, M. Pirotta, C. D'Eramo, and M. Restelli, "Boosted fitted q-iteration," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70. PMLR, 2017, pp. 3434–3443.

[62] S. Verma, J. Fu, S. Yang, and S. Levine, "CHAI: A chatbot AI for task-oriented dialogue with offline reinforcement learning," in *Proceedings of the North American Chapter of the Association for Computational Linguistics, NAACL*, 2022.

[63] S. Wang, X. Chen, D. Jannach, and L. Yao, "Causal decision transformer for recommender systems via offline reinforcement learning," 2023.

[64] A. Warnecke, L. Pirch, C. Wressnegger *et al.*, "Machine unlearning of features and labels," in *30th Annual Network and Distributed System Security Symposium, NDSS*, 2023.

[65] Y. Wu, J. McMahan, X. Zhu *et al.*, "Reward poisoning attacks on offline multi-agent reinforcement learning," in *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI*, 2023, pp. 10 426–10 434.

[66] H. Xu, T. Zhu, L. Zhang, W. Zhou, and P. S. Yu, "Machine unlearning: A survey," *ACM Comput. Surv.*, vol. 56, no. 1, pp. 9:1–9:36, 2024.

[67] H. Yan, X. Li, Z. Guo *et al.*, "ARCANE: an efficient architecture for exact machine unlearning," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI 2022*, pp. 4006–4013.

[68] Y. Yang and ufuk topcu, "Value-based membership inference attack on actor-critic reinforcement learning," 2023. [Online]. Available: https://openreview.net/forum?id=wKIxJKTDmX-

[69] J. Yao, E. Chien, M. Du, X. Niu, T. Wang, Z. Cheng, and X. Yue, "Machine unlearning of pre-trained large language models," 2024.

[70] D. Ye, T. Zhu, C. Zhu, D. Wang, S. Shen, W. Zhou *et al.*, "Reinforcement unlearning," *arXiv preprint arXiv:2312.15910*, 2023.

[71] X. Zhan, H. Xu, Y. Zhang, X. Zhu, H. Yin, and Y. Zheng, "Deepthermal: Combustion optimization for thermal power generating units using offline reinforcement learning," in *AAAI*, 2022.

[72] L. Zhang, R. Zhang, T. Wu, R. Weng, M. Han, and Y. Zhao, "Safe reinforcement learning with stability guarantee for motion planning of autonomous vehicles," *IEEE Trans. Neural Networks Learn. Syst.*, 2021.

[73] Q. Zhang, J. Liu, Y. Dai *et al.*, "Multi-task fusion via reinforcement learning for long-term user satisfaction in recommender systems," in *The 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2022, pp. 4510–4520.

[74] Q. Zhang, C. Yang, J. Lou *et al.*, "Contrastive unlearning: A contrastive approach to machine unlearning," *arXiv:2401.10458*, 2024.

[75] X. Zhang, J. Sun, C. Gong *et al.*, "Mutual information as intrinsic reward of reinforcement learning agents for on-demand ride pooling," *arXiv preprint arXiv:2312.15195*, 2023.

[76] W. Zhou, S. Bajracharya, and D. Held, "Plas: Latent action space for offline reinforcement learning," in *Proceedings of the 2020 Conference on Robot Learning*, 2021, pp. 1719–1735.

Appendix

Due to space limitations, please refer to the full version [25] for more detailed appendices.

A. Investigated Tasks and the Dataset

We carry out experiments across three of MuJoCo's robotic control tasks (`Hopper`, `Half-Cheetah`, and

TABLE VII: Information of each task and the dataset.

Environments	Tasks	Chosen Datasets	Observations	Action Shape	Action Type	Data Size	Task Type
MuJoCo [60]	`Hopper`	"hopper-medium-expert"	11	3	Continuous	2×10^6	Robotic Control
	`Half-Cheetah`	"halfcheetah-medium-expert"	17	6	Continuous	2×10^6	Robotic Control
	`Walker2D`	"walker2d-medium-expert"	17	6	Continuous	2×10^6	Robotic Control

`Walker2D`) [60]. In the three robotic control tasks, there is a sensor to monitor the state of the game, collecting information about the robot. Information collected from the sensor is used to observe the agent. In particular, the observation in `Hopper` is a vector of size 11. Observation in `Half-Cheetah` and `Walker2D` is a 17-dimensional vector. These vectors record the positions, velocities, angles, and angular velocities of different components of a robot.

The datasets used for these tasks are sourced from D4RL [17], a recently introduced and mostly studied benchmark for evaluating offline RL algorithms. D4RL provides a variety of datasets for four tasks, gathered through diverse policies, including *medium*, *random*, *medium-replay*, and *medium-expert*. We select the dataset that yields the highest returns for each task. We summarize the overview of datasets investigated in our experiments in Table VII.

B. Principles in Offline Reinforcement Unlearning

Drawing upon principles from unlearning in supervised learning [64], [33], we define that an optimal offline reinforcement unlearning method must adhere to properties as follows.
Efficacy. The primary objective for successful offline reinforcement learning is to erase as much as possible the agent's memory of the target trajectories. As this paper pioneers the concept of an unlearning method for offline RL, there is no established criterion to verify the effectiveness of unlearning within this area. Consequently, Section IV introduces a metric designed to assess unlearning in offline reinforcement learning.
Fidelity. An unlearning method is valuable only when it retains the performance of unlearned agents closely aligned with the original agents. Our objective is that the unlearned agent will not experience significant performance degradation. This property evaluates how well the model retains knowledge of the trajectories that are not intended to be forgotten.
Efficiency. A direct unlearning approach involves retraining agents from scratch on a dataset that excludes the target trajectories. However, this approach entails substantial runtime and storage overheads. The ideal unlearning method should minimize computational resource usage while ensuring that the unlearned agents perform as the retrained agent.
Agent Agnostic. The optimal approach for implementing offline reinforcement unlearning should be universally applicable across agents trained with various algorithms, which means that the objectives mentioned above should be met by any agent performing that unlearning strategy.

C. Offline RL Algorithms

This section introduces six investigated offline RL algorithms in our experiments. We provide a detailed discussion of the offline RL algorithms in Appendix D of our full paper [25].

D. Supplementary experiments

1) Error-Based Auditing: This section investigates the variations in expected TD error (as described in Eq. (6)) between trajectories that are included in and excluded from the agent's training dataset. Thus, we can assess whether TD error is an effective auditing metric. TD error quantifies the difference between the expected reward that a model anticipates for a specific action in a given state and the actual reward it observes, in addition to the predicted reward for the next state. This prediction relies on the agent's current policy and value function. This error shares similar concepts with loss in supervised learning. The ideal assumption is that the TD errors of trajectories included in the tested agents' training dataset would be significantly smaller than those obtained from trajectories excluded from the tested agents' training dataset.

Appendix E.1 our full paper [25] presents that there is minimal difference in TD errors between trajectories included in the agents' training dataset and those excluded from it. When the trajectories in the unlearning dataset and remaining dataset are collected using different behavior policies (unlearning dataset from the 'expert' policy and remaining dataset from the 'medium' policy), a significant difference is observed.

As shown in Table X of the full version of our paper [25], the difference in TD errors may simply caused by the trajectories' collected policies. These results suggest that TD error may not be a suitable metric for auditing.

2) Fidelity evaluation of TRAJDELETER*:* Appendix E.2 our full paper [25] presents that agents unlearned through TRAJDELETER exhibit a performance level that closely matches that of agents retrained from the ground up. Specifically, the average cumulative returns reveal only a slight variance when comparing the unlearned agents using TRAJDELETER to those undergoing the retraining process. This finding implies that TRAJDELETER does not detrimentally affect the agents' operational effectiveness in actual environments, thus underscoring the practical viability of our method. When using the random-reward method for unlearning, there is an average performance decrease of 11.4% (we derived this figure by averaging the performance changes relative to those of retrained agents) in the agents, indicating that trajectories involving random rewards can detrimentally affect the performance of unlearned agents. Employing the fine-tuning method for unlearning agents results in only minimal performance degradation. However, as illustrated in Table II, fine-tuning fails to effectively eliminate the impact of target trajectories on the original agents. Consequently, the fine-tuning approach is not an optimal method for unlearning.

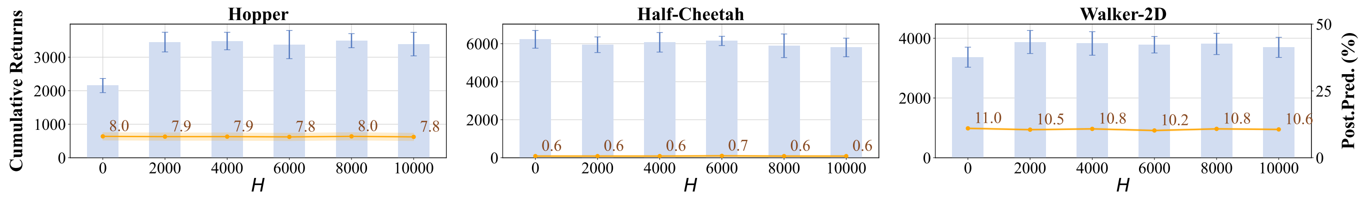

Fig. 7: The influence of convergence learning steps, H, on the unlearning performance of TRAJDELETER. Yellow points denote the average percentages of Positive Predictions (Post. Pred.) by TRAJAUDITOR for unlearned agents, aligning with the *left* y-axis. Shaded areas denote the standard deviation. Light blue bars illustrate the average returns, corresponding to the *right* y-axis, with the error bars indicating their standard deviation. The unlearning rate is 0.01.

3) Hyper-Parameter Analysis: As defined in Section VI-D, higher averaged cumulative returns indicate better overall utility of the unlearned agent, while a lower percentage of positive predictions means more complete forgetting of targeted trajectories. Appendix E.3 of Table XII of our full paper [25] illustrates the impact of forgetting learning steps K on the unlearning efficacy in TRAJDELETER. These results indicate a strong sensitivity of unlearning efficacy to the magnitude of forgetting learning steps K, suggesting that higher K values enhance unlearning effectiveness. Figure 7 also presents that with only 2000 steps of convergence training, the unlearned agents can recover their performance and achieve a level of average cumulative returns comparable to that of agents before unlearning. The cumulative returns and average percentages of Positive Prediction (Post. Pred.) remain stable for agents trained with an increasing number of convergence steps. Furthermore, the utility of the unlearned agents is largely unaffected by changes in K, implying that a higher K may be beneficial for the overall unlearning processes.

Table XIII of our full paper [25] delineates the influence of the balancing factor λ on TRAJDELETER's unlearning efficacy with a constant forgetting training step count of 8000. Most algorithms, excluding PLAS-P, exhibit nearly zero positive predictions, indicating robust unlearning efficacy that remains stable as λ increases. Conversely, when the forgetting training steps are set at 4000, as shown in Table XIV of our full version paper [25], a larger λ value leads to improved unlearning efficacy when the number of forgetting steps is small.

In summary, the forgetting learning steps K exert a more significant impact on unlearning efficacy compared to the balancing factor λ. A higher K enhances unlearning efficacy while maintaining the utility of the agent. Conversely, an increased λ value improves unlearning efficacy only when K is moderate, but it may negatively affect unlearned agents.

E. Additional Discussions

This section explores the robustness of TRAJAUDITOR and threats to validity, and the limitations of our paper.

1) Robustness of TRAJAUDITOR*:* Prior studies also explore the robustness of the auditing [13]: whether trajectory with some random noises can still be accurately identified as part of an agent's training dataset. For example, the MuJoCo environments also add small random noises to sensor information (i.e., observing states). Additionally, the offline RL agent deployed in real-world decision-making tasks that frequently utilizes Gaussian noise to improve the agents' generalization capabilities [1]. Thus, introducing Gaussian noise into the states of trajectories is a subtle approach to evade detection by an auditor. An effective trajectory auditing method expects to detect trajectories reliably, even if such noise exists.

We follow the MuJoCo [60] documentation to introduce Gaussian noise into each state of the trajectories. This noise follows a distribution with a mean of 0 and a standard deviation of 0.05. Our experimental results presented in Table VIII reveal that TRAJAUDITOR is only slightly affected by the Gaussian noise. When processing trajectories that have been perturbed with this noise, the average F1-score of TRAJAUDITOR decreases by both 0.02 across the three evaluated tasks compared to the performance on original trajectories. This consistency in performance highlights the robustness of TRAJAUDITOR to maintain accuracy and reliability under varying conditions.

2) Threats to Validity: To reduce the impact of environmental randomness and enhance the construct validity, we train the agents using five different random seeds, and each agent is allowed to interact with the environment to generate 100 trajectories. We then use the average cumulative returns from these trajectories to measure the agent's performance. The findings presented in this paper might have limited applicability to other offline datasets or algorithms. Our experiments are performed using the dataset from the benchmark recently introduced in [17], as well as six advanced offline RL algorithms, to relieve the threats to external validity.

3) Effectiveness Uncertainty.: We agree that, as proposed in TRAJDELETER, poor performance on specific trajectories cannot ensure that the agents have forgotten the trajectories. It is noticed that as an approximate method, the unlearning effect relies more on empirical evaluations; this is a common practice. In addition, the evaluation of TRAJDELETER depends on the effectiveness of the trajectories auditing method, i.e., TRAJAUDITOR. This issue is also a common challenge in previous unlearning research across various fields [64], [66]. We propose potential solutions by enhancing the auditor tools. Improving the auditor tool enhances unlearning methods and increases their trustworthiness. We remain open-minded about this issue, emphasizing that our conclusions are based on the presented effectiveness of TRAJAUDITOR.

4) Towards Legal Requirements: We focus on approximate unlearning. We summarize challenges and limitations that need further exploration to satisfy legal requirements.

TABLE VIII: The relative changes in precision, recall, and f1-score of TRAJAUDITOR after perturbing the tested trajectories, when compared with its performance on unaltered trajectories.

Tasks	Algorithms	Precision	Recall	F1-score
`Hopper`	BEAR	-4.0 %	-0.5 %	-0.02
	BCQ	-2.7 %	0.0 %	-0.01
	CQL	-5.9 %	0.0 %	-0.03
	IQL	-4.9 %	0.0 %	-0.01
	PLAS-P	-3.0 %	-1.2 %	-0.01
	TD3PlusBC	-9.1 %	0.0 %	-0.05
	Average	**-4.9 %**	**-0.3 %**	**-0.02**
`Half-Cheetah`	BEAR	+12.0 %	-13.8 %	-0.02
	BCQ	-1.3 %	-1.5 %	-0.01
	CQL	-3.0 %	+2.0 %	-0.01
	IQL	+2.9 %	+1.6 %	+0.02
	PLAS-P	-3.9 %	0.0 %	-0.02
	TD3PlusBC	-6.0 %	-0.5 %	-0.04
	Average	**+0.8 %**	**-2.0 %**	**-0.02**
`Walker2D`	BEAR	-1.8 %	-1.8 %	-0.02
	BCQ	0.0 %	0.0 %	0.0
	CQL	-0.3 %	-1.9 %	-0.01
	IQL	+0.4 %	-1.8 %	-0.01
	PLAS-P	+0.6 %	-4.7 %	-0.03
	TD3PlusBC	-1.5 %	0.0 %	-0.01
	Average	**-0.4 %**	**-1.7 %**	**-0.02**

- **Adequacy of Data Removal**: Approximate unlearning may not effectively remove all influence of an individual's data from the model as GDPR and other data protection regulations require, potentially leaving residual data that could still affect the model's output. We need to verify and validate the effectiveness of data unlearning methods.
- **Legal Acceptance**: Legal standards require both transparency and demonstrable efficacy; thus, developers of approximate unlearning techniques need to ensure that their methods are transparent and understandable to regulators.
- **Evolving Legal Standards**: Changes in data protection laws could impact the acceptance of approximate unlearning methods. Legal reforms or clarifications could either facilitate or hinder the adoption of these methods.

Despite these uncertainties, approximate unlearning reduces computational overhead and enables more dynamic data management practices, making it a valuable research area. Ensuring that approximate unlearning meets the same standards as full retraining is challenging. We have made every effort to address this issue in our paper. We also recommend developing certified unlearning methods to advance practical unlearning applications. Interdisciplinary research that includes legal experts and technologists is crucial for navigating the compliance landscape and crafting approximate unlearning methods that are technically effective and legally robust.

F. Related works

1) Offline RL for Real-World Applications: Recently, offline RL systems work brilliantly on a wide range of real-world fields, including healthcare [46], [15], energy management systems [71], autonomous driving [72], [28], and dialog systems [62], [56]. In healthcare, online RL is not suitable, as it is ethically and practically problematic to experiment with patients' health. Thus, Mila et al. [46] used advanced offline RL methods to develop a policy for recommending diabetes and sepsis treatment optimization. Meanwhile, Emerson et al. [15] applied offline RL to determine the optimal insulin dose for maintaining blood glucose levels within a healthy range. In various areas, using existing data to learn a policy proved significantly more efficient than online RL methods [71], [72]. In energy management, Zhan et al. [71] introduced offline RL algorithms aimed at refining the energy combustion control strategy for thermal power generating units. An autonomous driving, researchers gather diverse driving behaviors from multiple drivers and then train the planning algorithm using offline RL methods [72], [28], [42].

2) Machine Unlearning: Deep machine unlearning [5], [7], [64], [59] refers to eliminating the knowledge of specific data point(s) on the already trained Deep Neural Networks (DNNs). This concept gains particular significance in privacy and data protection legislation, such as the European Union's General Data Protection Regulation (GDPR) [20], which mandates a "right to erasure." Deep machine unlearning is categorized into two main groups: exact unlearning [5], [29], [67] and approximate unlearning methods [59], [23], [22], [11]. In exact unlearning, the most straightforward approach is retraining the DNN from scratch thought, excluding the data that is requested to be forgotten from the training set. This is computationally intensive, especially considering DNNs are typically trained on large datasets [53]. Bourtoule et al. [5] introduced the SISA method for training DNNs by dividing the dataset into non-overlapping shards, thereby diminishing the necessity for complete retraining since the DNN can be retrained on just one of these shards. Leveraging the one-class classifier, Yan et al. [67] developed a method that accelerates the SISA while also ensuring the accuracy of the retrained model.

Unlike exact unlearning guaranteeing that the outputs of an unlearned DNN and a fully retrained DNN are indistinguishable, approximate unlearning aims to estimate the parameters of DNNs in a manner analogous to retraining the network from scratch [66]. Warnecke et al. [64] established changes in the training dataset to closed-form updates of the DNN parameters, enabling direct adjustments to the DNN parameters in response to unlearning requests. However, this method is only applicable to tabular data. Golatkar et al. [23], [21] presented an approximation of the training process based on the neural tangent kernel and used it to predict the updated DNNs parameters. Unrolling SGD [59] operated approximate unlearning by direct stochastic ascent using unlearned data.

3) Certified Unlearning: A range of recent works studied certified unlearning definitions [32], [34]. Certified unlearning offers the theoretical guarantee that the unlearned model is indistinguishable from a DNN retrained from scratch on the remaining dataset. Besides, for approximate unlearning, Guo et al. [32], and Sekhari et al. [51] have proposed methods to ensure that the distributions of an unlearned model and a retrained model are nearly indistinguishable, using differential privacy techniques. Warnecke et al. [64] introduced a certified unlearning scheme for systematically removing learned features and labels from models.

Artifact Appendix

Abstract. This artifact contains the implementations for the paper "TRAJDELETER: Enabling Trajectory Forgetting in Offline Reinforcement Learning Agents" [25]. It introduces how to conduct experiments on six offline RL algorithms and three tasks to demonstrate that TRAJDELETER requires only about 1.5% of the time needed for retraining from scratch. It effectively unlearns an average of 94.8% of the targeted trajectories yet still performs well in actual environment interactions after unlearning.

A. Description & Requirements

1) *How to access*: Our code can be pulled from the public repository on GitHub. Please refer to the link: https://github.com/2019ChenGong/TrajDeleter.

2) *Hardware dependencies*: The training of all offline RL agents is conducted on a server configured with Python 3.7.11, equipped with one NVIDIA GeForce A6000 GPU and 512GB of memory. Actually, our experiments only require a GPU with more than 4GB of memory.

3) *Software dependencies*: Our artifact requires the installation of MUJOCO[3], D3RLPY[4], and D4RL[5].

B. Artifact Installation & Configuration

For installation, please refer to the link: https://github.com/2019ChenGong/TrajDeleter/README.md. We elaborate on each step in this "README.md" file. The training of all offline RL agents is conducted on a server configured with Python 3.7.11. We provide a step-by-step guide on how to install our repository using both Anaconda and Docker images. We conduct experiments on three offline datasets, "hopper-medium-expert-v0", "halfcheetah-medium-v0", and "walker2d-medium-v0". This repository can be easily extended to other datasets in D4RL. Besides, users can also utilize other algorithms implemented in D3RLPY[4].

The codes and scripts for replicating our experiments can be found in "README.md" under the "unlearning" folder.

C. Experiment Workflow

Firstly, please download the model from Google Drive: https://drive.google.com/drive/folders/1MeGkaGAZa_NXJUuk7GhfzyS_bsUHm8Z3, and please move these folders to the "unlearning" folder. This link provides access to three types of models: (1) Shadow Models, shadow agents used for auditing; (2) Exact Unlearning Agents, agents retrained from scratch; and (3) Fully Trained Agents, agents trained using the entire dataset.

After installation, please run python "env_test.py" to download and test the offline dataset. We describe the process for obtaining the experimental results in our paper as follows:

- For training the original agents, the hyper-parameters settings of offline RL algorithms are recorded in fold "./params". Please run the code: "python mujoco_fully_training.py". After training, the trained models are saved into the folder "./Fully_trained/ $<$ dataset_name $>$". You can download the well-trained original agents from our provided link.
- For Table 2 of RQ1, please run the code: "python script-auditor.py". After auditing, the results are saved into the folder "./ $<$ output_csv $>$".
- For Table 3 of RQ2, the agents used for the unlearning experiments in the "./Fully_trained/ $<$ dataset_name $>$" folder. The weights of the agents are named "model.pt", and the hyper-parameters settings of the offline RL algorithm are named ".json". Please run the code: "python mujoco_trajdeleter.py". After unlearning, the unlearned agents are saved into the folder "./Mujoco_our_method/ $<$ dataset_name $>$". Besides, you could run "python script-mujoco-trajdeleter.py" for all algorithms and tasks.
- For Table 4 of RQ2, if you want to test the performance of an agent, please run the code: "python performance_test.py". Besides, you could run "bash script-mujoco-test.py". To test the successful unlearning rates of the unlearned agents, please run the code: "python script-trajauditor.py".
- For RQ3 hyper-parameters analysis, you can edit line 58 and line 71 to change the "stage1_step" and "lamda" variable, which controls the number of steps for unlearning and the balancing factor, to obtain the experimental results.
- For defending against trajectory poisoning, please run "bash script-poisoning.sh" to poison the agents. This script is designed to poison the agents across all datasets and agents. Additionally, you can run "python poisoning_training.py" to target specific algorithms and tasks.
- To retrain the agents on a clean dataset and obtain agents without any poisoning, please run "bash script-poisoning-retain.sh".

For more details on the code, we highly recommend readers refer to our GitHub repository.

RL agents usually face the notorious problem of unstable performance. To reduce the impact of environmental randomness and enhance the construct validity, we train the agents using five different random seeds, and each agent is allowed to interact with the environment to generate 100 trajectories. We then use the average cumulative returns from these trajectories to measure the agent's performance. Therefore, it is normal for the results to fluctuate across different experiments, even when using the same experimental settings.

D. Copyright

We have migrated the code repository from GitHub to Figshare. The DOI for the repository is "10.6084/m9.figshare.26928706".

E. Acknowledgement

We thank the authors of D3RLPY[4] for providing the code for the offline RL algorithms and D4RL[4] for the offline datasets used in our evaluations.

[3]https://github.com/google-deepmind/mujoco/releases/tag/2.1.0
[4]https://github.com/takuseno/d3rlpy
[5]https://github.com/Farama-Foundation/D4RL