PERU: Policy Editing for Reinforcement Unlearning

Manuel Fabiano*, Francesco Longo*, Giovanni Merlino* and Francesco Restuccia†
*University of Messina, Italy †Northeastern University, United States
Emails: manuel.fabiano@studenti.unime.it, {francesco.longo, giovanni.merlino}@unime.it, f.restuccia@northeastern.edu

***Abstract*—Reinforcement learning agents deployed in production systems may later be required to remove the influence of specific training trajectories: to comply with privacy regulations such as GDPR, to roll back the effect of poisoned data discovered after deployment, or to align the agent with updated operational rules. Unlike supervised models, where unlearning a sample affects only its own prediction, an agent that has unlearned a trajectory continues to operate in the environment, and the behavior it adopts on the affected states is itself a security-relevant property. Existing reinforcement unlearning approaches leave that behavior undefined. We propose Policy Editing for Reinforcement Unlearning (PERU), a framework for deployed agents that treats unlearning as behavior replacement: the data owner specifies what the agent should do on the affected states, while an auditor verifies divergence from the recorded behavior. PERU operates directly on the policy network through a two-term objective that steers it toward the specified behavior on the forget states while preserving its original output elsewhere. We evaluate PERU on Atari and MuJoCo across seven online and offline RL algorithms, under an audit protocol adapted for policy-based unlearning, and against retraining and fine-tuning baselines. PERU raises the divergence from the recorded actions on forget trajectories to 99.0% while leaving it at 2.2% on retain trajectories, and repairs an agent compromised by a published poisoning attack using corrective actions for 0.83% of the poisoned transitions.**

I. Introduction

Once a reinforcement learning agent is in operation, its owner may be required to remove the influence of part of its training data. The request may come from a privacy regulation, such as the right to be forgotten established by the European Union's GDPR [1] and California's CPA [2], from the discovery, after deployment, that some of the training trajectories were poisoned, or from a change in operational rules the agent must now follow. In all three cases the request is framed as removing influence, and the owner needs it done without retraining the agent from scratch and without disturbing its behavior elsewhere, which is the problem machine unlearning addresses [3], [4].

Removing a sample from a classifier affects that sample's prediction and nothing else. An RL agent, by contrast, keeps acting: it will encounter the affected states again, and it will do something there. What it does is the property that matters, because an agent whose behavior on those states is undefined is a liability wherever it remains deployed. The owner therefore needs an answer to two questions: what the agent should forget, and what it will do instead.

Existing reinforcement unlearning methods answer only the first, and the reason lies in where they act. One removes the influence of an entire training environment, through mechanisms that require continuous environment access [5]; the other operates on the critic, so the change reaches the acting network only indirectly and the method makes no claim about what the agent should do on the affected states [6]. Verification inherits the same problem, since the established evidence asks whether the removed data can still be detected, not what the agent now does in its place.

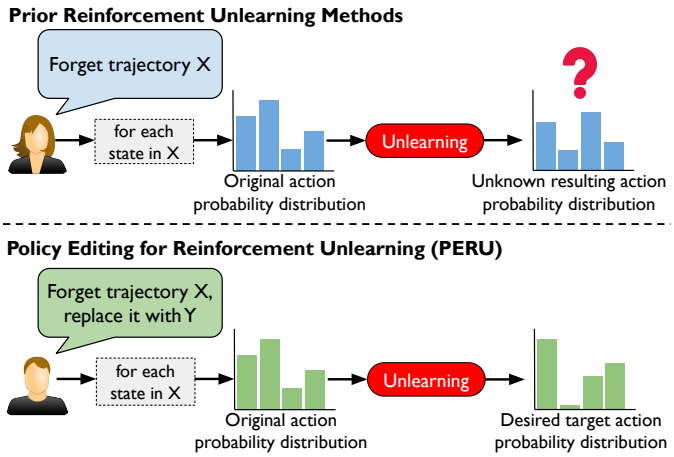

Fig. 1. PERU vs Existing work.

This paper treats reinforcement unlearning as behavioral substitution rather than retraining-equivalent removal. For deployed actor–critic agents, the operational effect of an edit is expressed through the policy outputs, since the critic is not queried at inference time. We therefore edit the policy directly and audit its behavioral divergence.

We realize this as Policy Editing for Reinforcement Unlearning (PERU). Figure 1 contrasts it with prior methods: where they leave the resulting action distribution unspecified, PERU steers it toward one the owner chose. Given a deployed agent, the owner specifies a set of state–action pairs to forget, a trajectory or any finite set identified by an audit, and optionally a replacement action for each. PERU then edits the policy through a two-term objective: one term steers it toward the replacement on the forget states, the other holds it to its original output on a retain set that represents its behavior

Network and Distributed System Security (NDSS) Symposium 2027
22 - 26 March 2027 , Seoul, Republic of Korea
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.[23|24]xxxx
www.ndss-symposium.org

elsewhere. Because the replacement is specified, target fidelity can be measured directly, while the auditor tests whether the original behavior is no longer reproduced. Constructing the retain set is the part that depends on what remains available at the time of the request, and PERU supports three regimes, from an accessible environment to a frozen replay buffer to nothing but the policy network itself; the optimization is the same in all three.

Summary of Contributions

• We formulate reinforcement unlearning as behavior replacement rather than removal, under a threat model in which the deployed artifact is the policy: a data owner specifies the desired behavior on the affected states and an auditor tests divergence from the recorded behavior. We propose PERU, which edits the policy directly through a two-term objective and supports three data-access regimes for retain set construction, from online environment access to a setting where only the policy network is available.

• We evaluate PERU on Atari and MuJoCo across seven online and offline algorithms, measuring *target fidelity*, the fraction of the distance to the specified behavior that the edit covers, and *behavioral preservation*, the complement of the drift from the original policy elsewhere. Behavioral preservation stays above 90% in almost every configuration, and we report the ones in which target fidelity falls short. Under an audit protocol adapted from TrajAuditor [6], PERU reaches a divergence of 99.0% on forget trajectories while leaving it at 2.2% on retain trajectories; the retraining and fine-tuning remedies evaluated either fail to install the target or incur severe utility loss when enforcing it.

• We repair an agent compromised by a published data-poisoning attack using corrective actions for 0.83% of the poisoned transitions, reducing the attacker's control on triggered states never poisoned from 0.979 to 0.099, against 0.261 for an agent retrained from scratch on clean data. We also show that four parameter-based auditors fail to recover a membership signal in this regime: they fail to separate policies trained with the forget set from policies trained without it, and the gap they would need does not grow when that influence is amplified tenfold.

II. Related Work

A. Machine Unlearning in Supervised Settings

Machine unlearning was introduced as removing the influence of specific training samples without full retraining [7]. Formal frameworks define it as distributional indistinguishability from a model retrained on the retain set [8], [9], achievable for restricted model classes [10]; deep networks trade that guarantee for practicality, by partitioning the data [11] or by approximate updates [12], [4], and verification then rests on membership inference. None of these extends naturally to reinforcement learning, where data is sequential, non-i.i.d., and generated by the policy itself.

B. Membership Inference and Influence

Membership inference attacks test whether a data point was used in training and are the standard empirical auditing tool for privacy and unlearning [13], [14]. In supervised unlearning, low accuracy on forget samples is read as evidence that their influence was removed. The statistics range from the training loss [15] to influence functions [16]. For reinforcement learning, *TrajAuditor* [6] lifts the protocol to whole trajectories by aggregating per-state evidence from Q-value distributions. We build on TrajAuditor but operate in policy-output space, which both extends the protocol to methods that modify the policy and changes what it establishes (Section VI-D).

C. Reinforcement Unlearning

To the best of our knowledge, two prior works propose formal definitions of reinforcement unlearning, in mutually exclusive settings. Ye et al. [5] consider agents trained on multiple environments and define unlearning as removing the influence of one of them; both their mechanisms rely on continuous environment access, which rules out single trajectories and fully offline regimes. *TrajDeleter* [6] operates offline on the training dataset, minimizing Q-values on the forget trajectories and re-optimizing the policy on the remainder. Because it edits the critic without specifying the desired behavior on forget states, the policy there is shaped by the global optimization rather than by the owner: the behavior is unconstrained, neither specified nor verifiable by an auditor.

III. Problem Setup and Notation

A. Markov Decision Process

Reinforcement learning problems are typically modeled using a Markov Decision Process (MDP) [17], which is defined as $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R \rangle$, where $\mathcal{S}$ and $\mathcal{A}$ are the set of states and actions respectively, $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the state transition probability function, where $P(s'|s, a)$ denotes the probability of transitioning to state s' from state s after taking action a, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function. The agent learns a policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, where $\pi(a|s)$ [18] represents the probability of taking action a in state s. During training, the agent collects finite trajectories, which are sequences of transitions:

$$\tau = \{(s_0, a_0, R(s_0, a_0), s_1), \ldots, (s_t, a_t, R(s_t, a_t), s_{t+1})\}.$$

that are used to update the policy π to maximize the expected cumulative reward.

B. Definition of Reinforcement Unlearning

Let us assume that an agent's policy π_{old} has been trained to near–optimality on a task. Once deployed, the agent generates a behavioral footprint consisting of trajectories where the action taken in state s is sampled from $\pi_{\text{old}}(\cdot|s)$ ($a \sim \pi_{\text{old}}(\cdot|s)$). We denote by $D_f = \{(s_f, a_f)\}$ the finite set of state–action pairs whose associated behavior must be removed or altered (we will call it *forget set*), and by $\mathcal{S}_f = \{s_f : (s_f, a_f) \in D_f\}$ the corresponding set of states. In the most common case D_f is extracted from a trajectory τ_f that must be removed, so

that $D_f = \{(s_f, a_f) : (s_f, a_f, R(s_f, a_f), s_{f+1}) \in \tau_f\}$; but it may equally be identified by a property the affected states share, as when an audit reveals that the agent misbehaves in a particular configuration. We make no assumption on how D_f was obtained, nor on whether its elements are temporally contiguous: it may come from the original training buffer, from online interaction, or from an audit performed after deployment. In the simplest case, the one prior work addresses, the requirement is only to make every $(s, a) \in D_f$ unlikely under the new policy. More generally, we define a *target distribution* $\varphi(\cdot|s)$ for each state $s \in \mathcal{S}_f$, which represents the desired behavior in forget states after unlearning. Given a divergence measure $\mathrm{div}(\cdot\|\cdot)$ and tolerances $\varepsilon, \delta > 0$, a policy π_{new} is successfully edited if:

$$\begin{cases} \mathbb{E}_{s\in S_f}\Big[\mathrm{div}\big(\pi_{\text{new}}(\cdot \mid s) \,\|\, \varphi(\cdot \mid s)\big)\Big] \le \delta, \\ \mathbb{E}_{s\sim d^{\pi_{\text{old}}}}\Big[\mathrm{div}\big(\pi_{\text{new}}(\cdot \mid s) \,\|\, \pi_{\text{old}}(\cdot \mid s)\big) : s \notin \mathcal{S}_f\Big] \le \varepsilon. \end{cases} \tag{1}$$

The first inequality bounds the average divergence from the chosen target $\varphi(\cdot \mid s)$ on forget states, while the second bounds average drift elsewhere under the occupancy measure $d^{\pi_{\text{old}}}$[19]. The tolerances just formalize a successful unlearning; in Section V-C, we relax these constraints into a weighted loss function for practical optimization. Note that both conditions are stated as expectations, so they bound average behavior: a policy can satisfy them while leaving a minority of forget states unedited.

C. Definition of Access Regimes

Keeping the original behavior of an agent on every state except those explicitly marked for modification requires that we can sample trajectories representative of the pre-existing policy π_{old}. We denote any such collection of state–action pairs dictated by the policy as:

$$D_r \;=\; \big\{(s_r, a_r) \;:\; s_r \notin \mathcal{S}_f,\; a_r \sim \pi_{\text{old}}(\cdot \mid s_r)\big\},$$

and refer to it as the *retain set*. We denote with S_r the corresponding set of states. The availability of the retain set depends on what portion of the original interaction pipeline remains available at the time of unlearning. We identify three mutually exclusive scenarios:

- *Online.* The environment or simulator is available, so fresh roll-outs of π_{old} are possible.
- *Offline–Buffer.* The environment is no longer accessible yet a frozen replay buffer collected under π_{old} is available.
- *Offline–NoBuffer.* Neither the environment nor any replay buffer is available. The only inputs are the deployed agent itself, rough bounds on the state space, and the trajectory (or trajectories) to be forgotten.

Each regime imposes different constraints on the constructions of D_r, which are detailed in the following section; for the rest of the problem formulation, it's enough to remember that D_r is well-defined in all three regimes, although its fidelity and size vary significantly.

IV. Threat Model

We consider a deployed RL agent whose policy π_{old} was trained on trajectories collected from one or more users, and whose owner later receives a request to remove the influence of a subset of them, arising from a privacy regulation, from the discovery of poisoned trajectories after deployment, or from a policy update. In the poisoning case the trajectories were contributed by an adversary who controls neither the training process nor the model, but only part of the training data. Once unlearning is performed, the resulting policy π_{new} remains in operation.

a) Deployed Artifact: What a deployed agent exposes, and what it acts with, is its policy. For actor–critic algorithms the critic is not queried at inference time, so a trajectory's influence is observable only through the actions the policy takes. We therefore treat the policy as the artifact under consideration: `PERU` edits the actor and leaves the critic untouched, and both the target and the audit are defined on policy outputs. Deployments that query the critic at runtime, for instance for online safety verification, fall outside this model: there the critic is part of the deployed artifact too, and editing both networks is required. Section VIII-C3 discusses the extension. For value-based methods the distinction does not arise, since a single network serves as both.

b) Actors: The *data owner* has white-box access to the policy and to the forget set D_f, and depending on the regime of Section III-C may also have the environment, a replay buffer, or neither. Optionally the owner specifies a target $\varphi(\cdot|s)$ for each forget state; otherwise φ is constructed as in Section V-B. `PERU` does not establish that a target is safe or compliant, which is a property of the process producing it; what it provides is that the specified actions are installed within a tolerance δ, that the installation is verifiable, and that behavior elsewhere is preserved. The *auditor* (the data owner, a regulator, or a trusted third party) verifies whether unlearning was performed on a given trajectory. It queries π_{new} as a black box and has white-box access to π_{old}, which it needs to construct the shadow ensemble. It also holds the trajectories under examination together with their recorded actions, since auditing requires a reference for what should have been forgotten.

c) Security Goals: A successful procedure must satisfy three goals. *Behavioral controllability*: on the forget states π_{new} follows φ within a tolerance δ. Our formulation places this first, because under a policy-only threat model the behavior on those states is not a by-product of the removal but the property the owner is entitled to specify, and it is what distinguishes `PERU` from prior approaches. *Behavioral divergence certification*: on D_f, π_{new} no longer produces the recorded actions. We adopt verification by a strong auditor [12], [11] rather than the distributional indistinguishability of formal supervised unlearning [8], [9], which does not transfer to the policy-dependent state distributions of RL. *Utility preservation*: on the retain states π_{new} stays within a tolerance ε of π_{old}. The first and third are formalized by

Inequality (1) and implemented by the loss of Section V-C; the second is established by the protocol of Section V-E.

d) Assumptions and Scope: We assume the data owner acts in good faith and specifies targets reflecting legitimate requirements, and that the auditor applies the protocol correctly and reads its output as evidence about the deployed policy's behavior rather than as a probe for residual training data. The procedure is performed offline, before redeployment. Out of scope are external adversaries with only black-box access to π_{new}, since the audit is a tool for the owner rather than a privacy attack, and adversaries that interact with the unlearning procedure itself.

V. The PERU Unlearning Methodology

A. Retain Set Construction

The optimization objective introduced in the previous section (Inequality 1) steers the policy away from the forget set D_f while pulling it toward its original behavior on the retain set. D_r is generated depending on the access regime, as follows.

If the environment is available, we construct the retain set by rolling out the original policy π_{old}, filtering out any state too close to the forget set. To encourage broader state-space coverage, we optionally employ an ϵ-greedy exploration strategy: with probability ϵ, the agent executes a random action in the environment, but always logs the action dictated by π_{old} in the retain set.

If a frozen replay buffer $D = \{\tau_i\}_{i=1}^{N}$ generated during the original training is available, we can filter out any transition belonging to D_f and consider the remainder as $D_r = D \setminus D_f$. Although the buffer cannot be expanded, its coverage is usually sufficient: MuJoCo datasets from D4RL still contain $10^5 - 10^6$ visits to states outside the forget set even when D_f spans 10^3 pairs.

When neither the environment nor a replay buffer are available, the feasibility of a surrogate retain set depends on the *structure of the observation space*. For tasks where observations stay within a known range $\mathcal{O} \subseteq \mathbb{R}^d$ (e.g., MuJoCo joint positions and velocities), retain states can be obtained by performing task-agnostic perturbations, such as *(i) range sampling*: for every dimension j, we draw $s_j \sim \text{Uniform}\big([\underline{s}_j,\ \overline{s}_j]\big)$ using the empirical bounds $\underline{s}_j, \overline{s}_j$ logged during training; *(ii) neighborhood jitter*: for each forget state $s \in \mathcal{S}_f$, we add isotropic Gaussian noise and then clip the result back into the correct bounds $\mathcal{O}$: $s' = \text{clip}(s + \delta, \underline{s}, \overline{s})$. Finally, every synthetic state can be fed through the original policy and the pair $(s, \pi_{\text{old}}(s))$ is appended to the retain set D_r.

When $\mathcal{O}$ is a large pixel space, such as the $4\times84\times84$ stacks of Atari, per-pixel perturbations produce frames the encoder was never trained on, and we do not propose a general solution. For Pong we evaluate a task-specific heuristic instead: the play area is divided according to its three foreground objects, the ball and the two paddles, and new states are generated by applying horizontal and vertical shifts to these elements starting from the forget states. Section VI-F reports what this achieves.

B. Target Distribution Design

The effectiveness of PERU depends on the replacement distribution $\varphi(\cdot|s)$ the policy is asked to match on every forget state. This distribution plays two roles: it specifies what behavior should replace the original one, and it shapes the magnitude and direction of the gradients that perform the change. From a security perspective it is also the interface through which the data owner expresses intent.

The primary case is the one in which a replace dataset is supplied: for each forget state the owner specifies the action that should now be preferred, and the distribution of those actions is the target. This is the operationally relevant setting, since it is the one in which the replacement behavior carries meaning, and it is the setting of the case study in Section VII. When no such dataset is available, φ must be constructed algorithmically, subject to two requirements: it must stay far enough from the original policy for the gradients to be informative, and it must remain numerically well behaved so that the KL does not diverge. Appendix E gives the two constructions we use, one per action-space type, together with their comparison. Their purpose is to exercise PERU across the full range of control settings on standard benchmarks, not to produce behaviors that are meaningful in themselves: whether a given target is safe or desirable is a property of the task and of whoever specifies it, not of the mechanism.

One aspect of the continuous case bears on results reported later. Deterministic policy gradient algorithms such as TD3 output an action $\mu_\theta(s)$ with no variance component, so computing a KL divergence requires assigning an artificial variance σ^2_{curr} to the current policy and a possibly different σ^2_{tar} to the target. The ratio $\sigma^2_{\text{tar}}/\sigma^2_{\text{curr}}$ modulates how aggressively the optimization pushes the policy away from its original behavior, so these variances are hyperparameters of the optimization landscape rather than quantities the model learns. For actors that do output a variance, such as those of SAC, CQL and IQL, the learned value enters the divergence directly and no artificial one is introduced.

C. PERU Loss Function Design

We formulate the loss as the sum of two complementary components, i.e., a *forget term*, which forces the desired policy to change on the states to forget, and a *retain term*, which forces the policy to remain close to the original on other states. In the forget term, we focus on the set of state-action pairs to erase D_f, where the new policy should match the previously defined target distributions $\varphi(\cdot|s)$ for each $s \in \mathcal{S}_f$. We parameterize the policy as π_θ, where θ represents the neural network parameters. The forget loss $L_{\text{forget}}(\theta)$ then measures the average KL divergence [20] between the current policy and this target distribution across all forget states:

$$L_{\text{forget}}(\theta) = \frac{1}{|S_f|} \sum_{s \in S_f} \text{KL}\left(\pi_\theta(\cdot \mid s) \parallel \varphi(\cdot \mid s)\right).$$

As stated in the previous section, for value-based methods, we convert Q-values into distributions via softmax with temperature τ. For deterministic policies, we impose artificial Gaussian distributions with fixed variances to enable KL computation. A lower value of this loss indicates that the current policy has successfully converged toward the target behavior defined by $\varphi(\cdot|\cdot)$ on the forget states. As such, the success of reinforcement unlearning hinges on how the target distribution is defined, i.e., if the target is poorly chosen (e.g., too similar to the original policy), the agent may not sufficiently alter its behavior. On the other hand, an excessive target may cause instability or unintended consequences.

In contrast, the retain term constrains behavioral drift outside the forget set. For a set of retain states S_r, constructed as described above, we define the retain loss as the average KL divergence between the updated policy and the original one on these states:

$$L_{\text{retain}}(\theta) = \frac{1}{|S_r|} \sum_{s \in S_r} \text{KL}\left(\pi_\theta(\cdot \mid s) \parallel \pi_{\text{old}}(\cdot \mid s)\right),$$

which acts as a direct constraint to keep the policy similar to the original behavior on all other parts of the state space.

Our final objective combines these two components as

$$\mathcal{L}(\theta) = \lambda_f L_{\text{forget}}(\theta) + \lambda_r L_{\text{retain}}(\theta), \tag{2}$$

where the weights λ_f and λ_r balance the emphasis between enforcing unlearning on the forget states and preserving performance elsewhere.

D. PERU Optimization Algorithm

Algorithm 1 summarizes the procedure. Once the forget set D_f, the retain set D_r and the target distributions $\{\varphi(\cdot \mid s)\}_{s \in S_f}$ are fixed, the edit is a sequence of deterministic full-batch gradient steps on the objective of Equation (2), identical in all three access regimes: the only regime-specific operation, constructing D_r, is completed before the optimization begins, so everything that follows manipulates tensors already resident in memory. Because there is no mini-batch noise the loss drops rapidly and then levels off, and the loop stops when it fails to improve for k consecutive steps.

Algorithm 1 PERU Unlearning

Require: Parameters θ_{old}, forget states S_f, retain states S_r, target distribution φ, original policy π_{old}, weights λ_f, λ_r, patience k

$\theta \leftarrow \theta_{\text{old}}$
Initialize Adam optimizer with θ
best_loss $\leftarrow +\infty$, *stall* $\leftarrow 0$
for $t = 1$ **to** $T_{\max}$ **do**
  // evaluate current policy on both sets
  Compute $\pi_\theta(\cdot \mid s)$ for all $s \in S_f \cup S_r$
  // compute losses on full batch
  $L_{\text{forget}} \leftarrow \frac{1}{|S_f|} \sum_{s \in S_f} \text{KL}(\pi_\theta \,\|\, \varphi)$
  $L_{\text{retain}} \leftarrow \frac{1}{|S_r|} \sum_{s \in S_r} \text{KL}(\pi_\theta \,\|\, \pi_{\text{old}})$
  $\mathcal{L} \leftarrow \lambda_f L_{\text{forget}} + \lambda_r L_{\text{retain}}$
  $\theta \leftarrow \text{Adam}(\theta, \text{clip}(\nabla_\theta \mathcal{L}))$ // weights update
  if $\mathcal{L} <$ *best_loss* **then**
    best_loss $\leftarrow \mathcal{L}$, $\theta_{\text{best}} \leftarrow \theta$, *stall* $\leftarrow 0$
  else
    stall $\leftarrow$ *stall* $+ 1$
  end if
  if *stall* $\geq k$ **then**
    break // early-stop on plateau
  end if
end for
return $\theta_{\text{new}} \leftarrow \theta_{\text{best}}$

E. Behavioral Divergence Audit Protocol

To verify that the edit was performed, the auditor needs a protocol that operates on the deployed policy and produces a per-trajectory verdict. We adapt TrajAuditor [6], the only existing membership inference attack designed for trajectory-level unlearning verification in offline RL, and we are explicit about what we read from it. The usual inference, that if an auditor cannot determine whether a trajectory was used in training then its influence has been removed, is not one we make: Section VI-D shows that it is not supported here, nor by the alternatives available. What the protocol establishes is behavioral: whether the policy still reproduces the recorded actions on the trajectory under examination.

Given a trajectory τ to audit, TrajAuditor compares the audited policy against an ensemble of N shadow models known to have seen τ. For each policy it computes a per-state value vector $v = (v_1, \ldots, v_T)$ with $T = |\tau|$, takes the element-wise mean $\bar{v}$ of the shadow vectors, and forms the Wasserstein distances $d_{\text{target}} = W(\bar{v}, v_{new})$ and $\{d_i = W(\bar{v}, v_i^{\text{shadow}})\}_{i=1}^N$. Grubbs' outlier test [21] is then applied to d_{target} against $\{d_i\}$: if the audited policy's distance is an outlier at significance level α, the trajectory is flagged as diverged. The test aggregates per-state evidence into a trajectory-level verdict.

Two adaptations are required. TrajAuditor builds shadow models by fine-tuning on the offline dataset, which presumes a static training set that online RL does not have; we instead construct a local shadow ensemble by adding calibrated Gaussian noise of scale σ_{shadow} to π_{old}, which is known to reproduce τ, which reuses the auditor's white-box access and applies in all three access regimes. And TrajAuditor sets v_t from Q-values, which would not reflect an edit confined to the policy network; we set $v_t = \|\pi(s_t) - a_t\|_2$, the distance between the policy's action and the recorded one, so that a small value means the policy still produces the recorded action on s_t. This formulation applies to any policy-based unlearning method.

The noise scale is calibrated to satisfy two conditions: the shadow ensemble must be diverse enough for the outlier test to be statistically meaningful, and before the edit every forget trajectory must be identified as remembered by the original policy, which validates the calibration. The protocol then produces a binary verdict per trajectory, aggregated into the *original-action divergence rate*: the fraction of audited trajectories on which the policy no longer produces the recorded

actions. On the forget set a high rate indicates that the specified change took place; on the retain set a low rate indicates that behavior there was preserved.

VI. Experimental Evaluation

A. Experimental Setup

We evaluate PERU on three MuJoCo locomotion tasks (Walker2d, Hopper, HalfCheetah [22]) and three Atari games (Pong, Enduro, SpaceInvaders [23]), accessed through the OpenAI Gym API [24]. Together they cover the settings of Appendix A1: continuous actions with low-dimensional observations, and discrete actions with pixel observations. On Atari we use DQN [25] and PPO [26]. On MuJoCo we use TD3 [27], SAC [28] and continuous PPO as online algorithms, and IQL [29], CQL [30] and TD3+BC [31] as offline algorithms trained on the D4RL medium-expert datasets [32]. We exclude continuous control algorithms that do not maintain an explicit policy network at inference time, since PERU operates directly on policy parameters.

1) Metrics: Three quantities correspond to the three security goals of Section IV, and a fourth reports task performance.

Target fidelity measures behavioral controllability: the fraction of the distance to the specified target that the edit covers, averaged over the forget states,

$$\mathrm{TF} = \frac{1}{|S_f|} \sum_{s \in S_f} \left(1 - \frac{d\big(\pi_{new}(s), \varphi(s)\big)}{d\big(\pi_{old}(s), \varphi(s)\big)} \right),$$

where d is the Euclidean distance between probability vectors in discrete control or between mean action vectors in continuous control. A value of one means the target was reached; zero means the policy did not move from where it started. The denominator is the distance the edit had to cover, so the quantity is comparable across targets of different magnitude.

Behavioral preservation measures utility preservation in action space: the complement of the normalized drift from the original policy on the retain states,

$$\mathrm{BP} = \frac{1}{|S_r|} \sum_{s \in S_r} \left(1 - \frac{d\big(\pi_{new}(s), \pi_{old}(s)\big)}{D_{\max}} \right),$$

with $D_{\max}$ the largest distance the output space admits, $2\sqrt{d_a}$ in normalized continuous action spaces where d_a is the action dimensionality, and the largest L_2 distance between one-hot vectors in discrete ones.

The *original-action divergence rate* is the output of the audit protocol of Section V-E: the fraction of audited trajectories on which the policy no longer produces the recorded actions. On the forget set a high rate means the specified change took place; on the retain set a low rate means behavior there was preserved. It is a per-trajectory verdict rather than an average over states, which is what makes it a distinct measurement.

Finally we report *cumulative reward* in the environment, since the three quantities above are defined in action space and a policy can score well on all of them while still losing performance: small per-state deviations compound over a trajectory and change which states are visited at all.

2) Relation Between the Metrics and the Objective: In behavior replacement the criterion of success is the specification itself: the data owner states what the policy should do on the forget states, and there is no ground truth external to that statement. Target fidelity therefore measures the same distance the forget term minimizes, and behavioral preservation what the retain term preserves. On those states a high value confirms that the optimization converged, and nothing more; the divergence rate follows from target fidelity, since the target is far from the recorded action by construction.

Two of our measurements do not have this property. The first is the same quantities computed on states the objective never saw: the held-out and unseen sets of Section VII-B, for which no corrective action was supplied, stand to the objective as a test set stands to a training loss. The second is cumulative reward, measured in the environment rather than in action space. The two are not equivalent: our ablations include a configuration that reaches a target fidelity above 99% while driving cumulative reward below zero, so satisfying the objective in action space is not sufficient.

3) Protocol and Reproducibility: Unless stated otherwise, the forget set consists of ten trajectories collected by rolling out π_{old} in the environment, and each unlearning experiment is repeated over ten runs with different forget trajectories, sampled by varying the environment seed. The retain set construction is made deterministic by fixing the random number generator and environment seeds before each roll-out, so that the collected retain states are reproducible. Edited policies are evaluated over 100 episodes with fixed seeds, on full-episode forget trajectories of approximately $1{,}000$ states. Hyperparameters are tuned per task–algorithm combination by grid search, once and reused across unlearning requests, which suits the operational scenarios of Section IV where unlearning is a rare event rather than a continuous workload; the search space is described in Appendix B. Throughout we use the opposite-torque target for continuous control and the ranking-preserving rule for discrete control, both defined in Appendix E. All experimental code, seed configurations and hyperparameter settings will be made available upon acceptance.

B. Behavioral Divergence under Audit

We evaluate the second of the security goals of Section IV by applying the audit protocol of Section V-E to policies edited by PERU and to those edited by TrajDeleter [6], the only existing reinforcement unlearning method directly comparable to ours. Both are audited under the same protocol. We focus the comparison on the offline algorithms (IQL, CQL, TD3+BC), where TrajDeleter was originally evaluated, across the three MuJoCo locomotion tasks; results on online algorithms, where TrajDeleter does not apply, are reported for PERU alone.

The two methods act on different networks, TrajDeleter modifies the critic and re-tunes the policy on the remaining data, without specifying any target behavior on forget states; under its original protocol, which inspects Q-values, that

TABLE I
ORIGINAL-ACTION DIVERGENCE RATE (%, MEAN WITH 95% CONFIDENCE INTERVAL) WHEN UNLEARNING TEN TRAJECTORIES. TRAJDELETER DOES NOT APPLY TO ONLINE ALGORITHMS.

		PERU (Ours)		TrajDeleter	
Task	**Algo**	Forget ↑	Retain ↓	Forget ↑	Retain ↓
H.Cheetah	TD3	100.0	0.0	–	–
	SAC	100.0	0.0	–	–
	PPO	100.0	18.2±5.2	–	–
	IQL	91.0±6.7	3.1±0.7	2.0±3.0	0.0
	CQL	100.0	0.0	30.0±16.2	14.6±9.4
	TD3+BC	100.0	1.3±0.3	99.0±2.3	28.6±12.3
Hopper	TD3	100.0	3.4±0.5	–	–
	SAC	95.0±5.1	16.9±6.7	–	–
	PPO	100.0	0.0	–	–
	IQL	100.0	0.6±0.4	73.0±13.9	26.3±3.8
	CQL	100.0	0.8±0.4	40.0±15.8	13.5±3.5
	TD3+BC	100.0	10.7±0.8	77.0±12.2	11.3±1.9
Walker2D	TD3	100.0	0.0	–	–
	SAC	100.0	0.0	–	–
	PPO	91.0±4.1	8.7±2.2	–	–
	IQL	100.0	0.0	87.0±8.3	4.8±1.5
	CQL	100.0	0.5±0.4	77.0±11.7	4.9±1.6
	TD3+BC	100.0	2.4±0.4	90.0±8.9	6.8±3.6
Avg. (offline only)		**99.0**	**2.2**	**63.9**	**12.3**
Avg. (all)		**98.7**	**3.7**	–	

modification is directly visible. Our audit inspects the policy instead, for the reason given in Section IV: a change confined to the critic is not a change in the deployed artifact. What Table I reports for TrajDeleter is therefore not the effect of its edit on the critic, but how much of that effect reaches the actor.

Table I reports the divergence rate on the forget and retain sets when unlearning ten trajectories. `PERU` reaches a forget divergence of 99.0% on average against 63.9% for TrajDeleter, a gap of 35 percentage points, and it does so while leaving the retain set almost untouched: 2.2% of retain trajectories are flagged as diverged, against 12.3%. The pattern holds across tasks and algorithms: `PERU` reaches complete divergence on the forget set in most offline configurations, whereas TrajDeleter leaves a substantial fraction of forget trajectories still producing the recorded actions, with lows of 2.0% divergence on IQL-HalfCheetah and 30.0% on CQL-HalfCheetah.

The propagation from critic to actor is therefore partial and uneven across tasks, and the run-to-run confidence intervals on the TrajDeleter columns are wide, reaching ± 16.2 percentage points. `PERU` edits the policy toward the target distribution directly, and that change is fully observable to a policy-level auditor.

C. Comparison with Alternative Remedies

The comparison with TrajDeleter concerns how much a critic-level method changes the policy. A separate question is how `PERU` compares against the procedures an operator would plausibly try instead. Table II reports three of them on ten forget trajectories drawn from the training dataset of `walker2d-medium-expert`, with the replacement behavior $\varphi(s) = -\mu_{old}(s)$. *Leave-out retraining* trains from scratch on the dataset with the ten episodes removed, while *retraining with target injection* replaces those episodes with R copies of the state–target pairs at a per-step reward of twice the maximum observed in the clean data, so that the injected behavior is the most rewarding transition the critic sees, and *fine-tuning with target injection* continues training π_{old} on that same dataset for 40% of a full training budget. Target fidelity is reported for every row, including those that specify no replacement behavior, since φ is defined independently of the method.

The two divergence columns are read together, since a method that changed the policy only where asked diverges on the forget trajectories and not on the retain ones, whereas one that produced a different policy altogether diverges on both.

Leave-out retraining changes nothing the audit can detect on any of the three algorithms, with both divergence rates at zero and target fidelity below 0.015: a policy retrained from scratch without the ten episodes still produces the recorded actions on them. The audit does discriminate, since the injected rows that install the target reach 100%. The retrained policies also perform less consistently: on CQL the return varies by ± 984 across evaluation episodes against ± 91 for π_{old}, and on TD3+BC by ± 806 against ± 59. The means differ too, though across independently trained policies that difference is within run-to-run variation. On IQL the ordering is reversed, with π_{old} at 4007 ± 1535 against 4980 ± 70, which illustrates the same point from the other side.

At the injection weights tested, stronger target enforcement sharply trades off target fidelity against task performance. At $R = 10$ the target is essentially ignored on CQL and TD3+BC, with fidelity 0.012 and 0.135, while at $R = 100$ it is partially reproduced, 0.787 and 0.635, at a cost of 96% and 87% of cumulative reward. IQL is the extreme of the same curve, since at the weight where CQL installs nothing it already reproduces 0.671 of the target and returns 41. Fine-tuning from π_{old} rather than from scratch does not change the picture, agreeing with retraining to within 0.02 of fidelity on every algorithm at comparable returns.

Two further procedures could be considered as baselines, but are not reported as such. Behavior cloning toward the replacement actions is not an alternative to `PERU` but a variant of it, obtained by substituting the divergences with squared errors, and appears as an ablation in Section VI-H. Weight-space regularization in the style of Elastic Weight Consolidation over-constrains the update in this setting, for the reason given in Section VIII-A: forget and retain states rely on shared internal representations, so preventing weight change prevents the edit.

D. Influence-Based Auditors

The protocol of Section V-E measures a behavioral property by design, and it is fair to ask whether the influence of a forget trajectory could be recovered some other way. For each of the three offline algorithms we train five policies on the full

TABLE II

ALTERNATIVE REMEDIES ON TEN FORGET TRAJECTORIES. ONE SEED; THE FORGET SET, THE RETAIN SET AND THE AUDIT REFERENCE ARE IDENTICAL ACROSS ROWS. div_f AND div_r ARE DIVERGENCE RATES ON THE FORGET AND ON RETAIN TRAJECTORIES. REWARD IS OVER TEN EPISODES.

	TD3+BC				CQL				IQL			
Method	div_f	div_r	fid.	reward	div_f	div_r	fid.	reward	div_f	div_r	fid.	reward
π_{old}	0	0	0.0	4803 ± 59	0	0	0.0	5108 ± 91	0	0	0.0	4007 ± 1535
Leave-out retrain.	0	0	0.1	4552 ± 806	0	0	0.5	4306 ± 984	0	0	1.4	4980 ± 70
Retrain., R=10	30	1	13.5	4948 ± 62	0	0	1.2	4021 ± 1647	100	72	67.1	41 ± 1
Retrain., R=100	100	97	63.5	614 ± 365	100	70	78.7	219 ± 28	100	87	79.3	383 ± 50
Fine-tun., R=10	10	0	13.4	42 ± 2	0	0	0.6	4965 ± 84	100	50	60.0	637 ± 279
Fine-tun., R=100	100	96	63.7	451 ± 146	100	71	77.7	184 ± 33	100	86	80.2	653 ± 196
PERU	100	2.4	97.8	4790 ± 39	100	0.5	98.1	4797 ± 32	100	0	97.1	4583 ± 54

dataset and five on the dataset with the forget set removed, using paired seeds. Four auditors then look for that influence in the parameters rather than in the actions.

The four statistics span three orders of the signal. At order zero we take the value of a behavior-cloning surrogate for the training loss, the canonical membership statistic: a sample the model was trained on should have lower loss [15]. At order one we take the norm of the gradient at the audited state–action pairs, and its cosine alignment with the mean gradient over retained data. At order two we take curvature-aware self-influence, evaluated through influence functions [16]. Each is computed on ten forget trajectories, which are 0.44% of the transitions of `walker2d-medium-expert`. Appendix C reports the estimation details, including why self-influence adds nothing to the gradient norm here.

Table III gives, for each statistic, the area under the ROC curve separating the two sets of policies. The *forget* column computes it on the forget trajectories, and the *control* column on held-out retain trajectories, which were used to train both sets and should therefore score 0.5. On a finite sample the control value scatters, and that scatter defines the null band; none of the sixteen values falls outside it. The most significant of the paired comparisons has $p = 0.106$, and correcting for the number of tests moves it further from significance.

A negative result of this kind is only informative if there is something to find, so we measured the memorization gap directly: the surrogate loss on the forget trajectories, relative to held-out ones, differs by 1.4% between the two sets of policies. We then retrained with the same ten trajectories duplicated tenfold, raising their share of the dataset to 4.2%; the gap does not grow and the auditors continue to find nothing. Two policies trained on the same task with datasets differing by ten episodes converge to similar behavior, even offline where the policy is constrained by its data. On those episodes their actions then differ too little for the statistics to separate them.

At this operating point, the behavioral audit therefore provides a usable verification signal where the influence-based auditors considered do not.

TABLE III

INFLUENCE-BASED AUDITORS ON TEN FORGET TRAJECTORIES, COMPUTED ON FIVE POLICIES TRAINED WITH THE FORGET SET AND FIVE WITHOUT IT. THE LAST BLOCK AMPLIFIES THE INFLUENCE OF THE SAME TRAJECTORIES TENFOLD. NO VALUE EXCEEDS THE UPPER EDGE OF THE NULL BAND.

Algorithm	Auditor	Forget	Control	p
Operating point: forget set is 0.44% *of transitions*				
TD3+BC	loss value	0.486	0.469	0.409
	gradient norm	0.431	0.485	0.831
	gradient alignment	0.369	0.426	0.935
	self-influence	0.437	0.483	0.792
IQL	loss value	0.475	0.511	0.997
	gradient norm	0.475	0.486	0.610
	gradient alignment	0.504	0.506	0.327
	self-influence	0.476	0.494	0.628
CQL	loss value	0.498	0.508	0.637
	gradient norm	0.487	0.485	0.582
	gradient alignment	0.402	0.432	0.740
	self-influence	0.493	0.495	0.437
Amplified: same trajectories at $10\times$ *multiplicity,* 4.2%				
TD3+BC	loss value	0.511	0.461	0.106
	gradient norm	0.440	0.504	0.938
	gradient alignment	0.513	0.500	0.582
	self-influence	0.429	0.491	0.849

E. Behavioral Controllability

Table IV reports target fidelity and behavioral preservation across all task–algorithm combinations, when unlearning one or ten trajectories. Online algorithms are evaluated in the online access regime, offline algorithms in the offline-buffer regime.

The target used throughout is the opposite-torque rule, $\varphi(s) = -\mu_{old}(s)$, which guarantees a distance of $2\|\mu_{old}\|$ from the original action and therefore exercises the mechanism with strong gradients. It is not a behavior anyone would want an agent to adopt. On states along the agent's operating path, installing it means asking the policy to do the opposite of what keeps the task going, so a loss of cumulative reward accompanies a successful edit rather than signalling a failed one. The two metrics in this table measure whether the

TABLE IV

TARGET FIDELITY AND BEHAVIORAL PRESERVATION (%) ACROSS ALL TASK–ALGORITHM COMBINATIONS, WHEN UNLEARNING 1 OR 10 TRAJECTORIES. HIGHER IS BETTER FOR BOTH. MEAN OVER TEN UNLEARNING RUNS; CONFIDENCE INTERVALS OMITTED AS THEY WERE NEGLIGIBLY SMALL.

Online RL - MuJoCo				Online RL - Atari				Offline RL - MuJoCo			
Algo	**Task**	**1T**	**10T**	**Algo**	**Task**	**1T**	**10T**	**Algo**	**Task**	**1T**	**10T**
TD3	HalfCh.	88.2/96.1	92.1/95.9	DQN	Pong	72.9/99.8	71.2/99.9	IQL	HalfCh.	90.6/98.0	78.8/97.8
TD3	Hopper	95.2/96.7	94.5/97.2	DQN	SpaceInv.	88.5/98.9	67.5/98.8	IQL	Hopper	88.1/96.0	68.0/93.1
TD3	Walker	91.8/95.0	90.1/94.9	DQN	Enduro	84.2/96.6	73.3/95.9	IQL	Walker	97.1/96.9	94.0/95.8
SAC	HalfCh.	84.5/92.6	79.3/92.9	PPO	Pong	79.8/88.7	71.3/94.6	CQL	HalfCh.	87.3/98.1	79.1/98.1
SAC	Hopper	59.1/96.1	55.2/96.8	PPO	SpaceInv.	81.9/98.7	77.8/98.4	CQL	Hopper	90.8/95.4	85.3/95.1
SAC	Walker	80.9/91.1	52.4/86.5	PPO	Enduro	92.1/98.4	67.7/97.7	CQL	Walker	98.1/97.2	97.1/96.9
PPO	HalfCh.	77.3/99.2	82.2/99.2					TD3+BC	HalfCh.	82.7/97.6	72.5/98.0
PPO	Hopper	67.9/98.4	69.5/98.5					TD3+BC	Hopper	80.3/96.6	71.0/96.5
PPO	Walker	35.7/95.1	10.7/96.1					TD3+BC	Walker	97.8/95.2	97.2/95.6

specified change was installed and whether behavior elsewhere was preserved; the cost in the environment is reported in Section VI-F.

Behavioral preservation stays at or above 90% in almost every configuration, and around or above 95% for PPO and TD3 on all MuJoCo tasks regardless of the number of forget trajectories. Target fidelity is where the algorithms differ. TD3 is the most stable, staying above 88% everywhere and losing at most 1.7 points between one and ten trajectories. Among offline methods, CQL and TD3+BC on Walker2D reach 97% with ten trajectories and IQL 94%. Across the twenty-four configurations the median loss from one to ten trajectories is 5.4 points.

Six configurations fall well outside that range. On PPO-Walker2D the edit barely takes hold, with a target fidelity of 35.7% at one trajectory and 10.7% at ten. SAC-Walker2D falls from 80.9% to 52.4% and is the only configuration where behavioral preservation also degrades appreciably, from 91.1% to 86.5%. SAC-Hopper stays near 57% at both sizes, so the difficulty there is not one of scale. DQN-SpaceInvaders, PPO-Enduro and IQL-Hopper each lose more than twenty points between one and ten trajectories. Four of the six use algorithms with stochastic policies, PPO and SAC, whose action distributions are broader and whose modes are correspondingly harder to move onto a point target, although the present experiments do not isolate that mechanism.

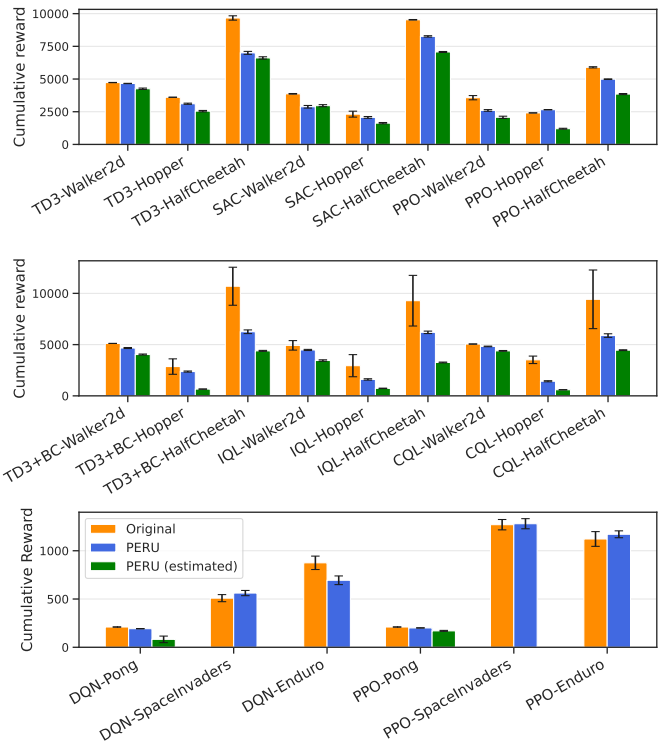

Fig. 2. Cumulative reward after unlearning one trajectory of 1000 states, on MuJoCo online (top), offline (center) and Atari (bottom). *Estimated* uses synthetic retain sets, when neither environment nor buffer is available.

F. Retain Set Accessibility

Figure 2 reports cumulative reward after unlearning one trajectory, with a valid retain set obtained from environment roll-outs or from a replay buffer, and without one, where retain states are generated synthetically from the policy network alone.

With a valid retain set the degradation is moderate, and part of it follows from the target rather than from the mechanism. The opposite-torque rule asks the policy to invert its action on the forget states, which costs return whenever those states lie on the operating path. Online algorithms on MuJoCo lose 14% on average with TD3, 17% with SAC and 11% with PPO. On Atari, Pong loses between 5% and 8%, while SpaceInvaders improves for both DQN and PPO, which suggests that the removed trajectory was not contributing to performance. Offline algorithms are more task-dependent: Walker2D stays within 5–9% across all three, while HalfCheetah loses between 33% and 42%.

Without a retain set the losses are larger throughout, and this is the clearest limitation of the method as currently formulated. Online algorithms lose less than offline ones, at 24–26% for TD3 and SAC and 42% for PPO. Offline algorithms fare worse: Walker2D loses between 13% and 30%, while Hopper loses 75–83% and HalfCheetah 53–65%, which leaves an agent that no longer performs the task. The synthetic construction samples from the observation bounds and from neighborhoods of the forget states, which are not on the

policy's occupancy measure. The retain term then constrains a distribution the agent does not visit, leaving the one it does visit free. Several of the worst configurations, notably CQL-HalfCheetah and IQL-HalfCheetah, also have high variance in the original policy, so training instability compounds the effect.

For Atari we evaluated only Pong in this regime, using the task-specific heuristic of Section V-A: DQN retains 39% of its original performance and PPO 81%. That heuristic exploits domain knowledge about Pong and does not transfer to other pixel domains.

G. Scalability in the Forget Set Size

The experiments so far edit one or ten trajectories. To characterize how the edit degrades as the forget set grows, we sweep $|D_f|$ over 1, 5, 10, 20, 50 and 100 trajectories on TD3+BC across the three MuJoCo tasks, with three repetitions per point. Again, the target is the opposite-torque rule throughout, which deliberately imposes a strong shift from the original action: a target closer to the original action would require a smaller change at every size.

Figure 3 reports target fidelity and cumulative reward, the latter as a fraction of the value the policy reached before editing. Across the configurations, behavioral preservation remains between 95.1% and 98.2%, and on every task it is marginally higher at 100 trajectories than at one. Preserving behavior in action space is, therefore, not what limits the edit at scale.

Target fidelity behaves differently by task. On Walker2D it is unchanged, from 97.8% at one trajectory to 97.3% at one hundred. On Hopper and HalfCheetah it drops by 8.4 and 11.0 points, but the drop happens between one and ten trajectories and then flattens: from ten to one hundred, Hopper moves from 69.9% to 72.1% and HalfCheetah from 72.0% to 72.9%, both upward.

Cumulative reward is not monotone in $|D_f|$ and separates the three tasks. Walker2D holds at 89–94% of its pre-edit value. HalfCheetah improves, from 56% at one trajectory to 83% at fifty, which is consistent with the forget term being averaged over more states and therefore pushing each one less hard, the same effect that lowers fidelity. Hopper degrades, from 84% to 48%.

The Hopper degradation admits a second explanation: the retain set is built by filtering roll-out states that lie within a fixed distance of the forget set, so a larger forget set removes more of them. On Hopper the retain set falls to 53% of its size at one trajectory, while on Walker2D and HalfCheetah it stays above 97%.

H. Design Choices

Appendix D reports four ablations: the forget-side divergence, the necessity of the retain term, the sensitivity to the three main hyperparameters, and the two target construction rules. Two of their outcomes are used elsewhere in the paper. Forward KL achieves the best trade-off among the divergences we tried, and the two that weight each action by its probability

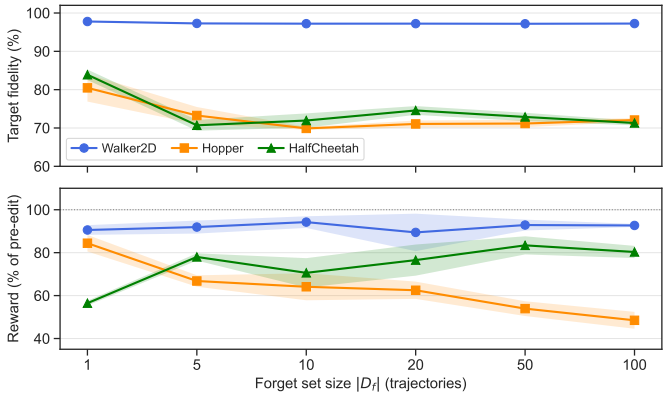

Fig. 3. Target fidelity and cumulative reward against forget set size, on TD3+BC. Reward is a fraction of the value reached before editing. Mean and standard deviation over three repetitions.

under the target rather than under the policy, reverse KL and Jensen-Shannon, produce almost no edit at all: their target fidelity is 2.9% and 5.3% against 80.9%. And the retain term is not optional: with $\lambda_r = 0$ the forget term alone reaches a target fidelity of 99.4% while driving cumulative reward to -4.3, that is, it destroys the policy everywhere.

VII. Case Study: Repairing a Backdoored Agent

The evaluation so far establishes that `PERU` installs a specified behavior and preserves the rest. This section asks whether that matters, on a deployed agent compromised by a published attack.

The attack is the data poisoning of BAFFLE [33] on `hopper-medium-expert`, reproduced from the misleading experiences released by its authors. The trigger overwrites three of the eleven observation dimensions, the torso's linear and angular velocities, with their per-dimension medians in the clean dataset; the action is replaced with one produced by a deliberately poorly performing agent, and the reward is set to 4.0, the third quartile of the clean reward distribution.

We apply the substitution at the transition level, to 10% of transitions sampled uniformly rather than to the last 10% of episodes as in the original implementation: in our reproduction the latter installs the backdoor on the poisoned states without generalizing beyond them, which leaves nothing for a repair to generalize to.

Three agents enter the comparison: the *poisoned* one, trained on the contaminated dataset; the *retrained* one, trained from scratch on the clean dataset; and the *repaired* one, obtained by editing the poisoned agent with `PERU`. Within an algorithm the three share configuration, number of gradient steps, and an observation scaler frozen to the statistics of the clean dataset, so that the only difference between them is the data they were trained on. The comparison covers TD3+BC, CQL and IQL over three training seeds each, with the poisoning and the evaluation sets generated once and reused, so that which transitions are contaminated and which states are audited is identical throughout.

Let's assume that, once the affected transitions have been identified, an external process provides validated corrective actions for 1,000 of them, 0.83% of the 120,092 poisoned transitions. No clean checkpoint is available for redeployment; were one available, redeploying it would be the simpler remedy. The corrective actions are the pre-contamination labels of those states, used here as a trusted external corrective set and as ground truth for evaluation. Their median distance from the actions the attacker installed is 1.685 on a maximum of 3.464, so the required edit is not marginal. Because those labels come from a mixture behavior policy they are not optimal, so a repaired agent matching them closely may perform slightly below the retrained one on those states.

A. Evaluation Criteria

The return obtained while the trigger is presented is not adopted here as the primary criterion. The trigger replaces three velocity channels with constants, and for a system that depends on velocity feedback to stay upright the effect is not specific to the poisoned agent: under a fixed presentation interval the retrained agent, which never saw a poisoned transition, loses 47.8% of its return and falls in 79 of 100 episodes. The degradation also depends on the presentation schedule in a way unrelated to the backdoor, since at other intervals the same agent loses under 3%.

What distinguishes a backdoor from sensory corruption is not that performance drops but that the output is controlled by the attacker, so the criteria used here measure that control directly. The *attacker-compliance rate* is the fraction of triggered states on which the policy's action is closer to the attacker's intended action than to the corrective one. *Trigger invariance* is $\|\pi(s^+) - \pi(s^-)\|$, the change the trigger induces in the output. It is not a term of PERU's objective, since the forget term drives $\pi(s^+)$ toward the corrective action while $\pi(s^-)$ is governed by the retain term, and the two coincide only if the corrective action is what the policy already does on the untriggered state.

The trajectory-level audit of Section V-E does not apply here, since the affected transitions are scattered across episodes rather than grouped into trajectories, and on a poisoned transition the recorded action is the attacker's, so divergence from it would establish only that the policy stopped obeying the attacker. Attacker compliance compares against both the attacker's action and the corrective one, and is therefore the two-sided version of the same measurement.

We evaluate three sets of states. The *edit* set is where corrective actions were supplied; the *held-out* set contains other poisoned transitions, for which none were; the *unseen* set contains triggered states that were never poisoned at all. The backdoor generalizes to the last of these, and whether the repair does is the question this section wants to answer. A fourth set of untriggered states serves as preservation control. Since the trigger degrades every policy, a successful repair is not one that restores performance under trigger, which no method can, but one that removes the attacker's control while remaining comparable to the retrained agent outside the edited states.

B. Results

Table V reports the three agents for the three algorithms, over three training seeds each. Compliance and invariance are medians over 1,000 states, and compliance on the *edit* set is governed by the objective itself and is therefore not independent evidence. Under the trigger the poisoned agent reproduces the attacker's action on all three triggered sets alike, including the states that were never poisoned, since the trigger was learned as a rule rather than memorized as a set of transitions. Without the trigger the agent behaves normally, so the defect is confined to an observable condition.

After editing with 1,000 corrective actions, attacker compliance on the *unseen* set falls from between 0.949 and 0.979 to 0.099 on TD3+BC, 0.109 on CQL and 0.103 on IQL, with a seed-to-seed spread of 0.008, 0.008 and 0.003. Trigger invariance falls from between 1.494 and 1.678 to 0.417, 0.502 and 0.448. The three algorithms therefore agree to within 0.010 on compliance and 0.085 on invariance: the repair generalizes to triggered states outside the poisoned set in the same way regardless of which algorithm produced the policy. The held-out and unseen figures agree as well, at 0.102 and 0.099 on TD3+BC, the repair therefore generalizes beyond the thousand edited instances.

The retrained agents reach 0.261, 0.198 and 0.191 on unseen compliance, with spreads of 0.025, 0.009 and 0.027, and between 0.789 and 0.885 on invariance. Both gaps between the repaired and the retrained agents are at least three times the seed-to-seed spread, and at least six times it on invariance.

The repair leaves untriggered behavior essentially unchanged, at 0.375 against 0.365 before editing on TD3+BC and comparable elsewhere. It costs little nominal return on the two stable algorithms: 3629 ± 8 to 3615 ± 14 on TD3+BC and 3622 ± 19 to 3603 ± 25 on CQL, four and five tenths of a percent. On IQL the cost is 4.3%, from 3210 ± 482 to 3072 ± 726, and the dispersion indicates why: the poisoned IQL agents fall in 47, 13 and 7 of 100 episodes with no trigger present, against at most 4 for the other two algorithms, and the repair starts from that state. On the edit set the residual distance to the corrective action is between 0.030 and 0.142 across the three algorithms, against 0.930 to 0.988 for the retrained agents, which is expected given the objective (Section VI-A2).

C. Comparison with Retraining

The retrained agents remove the influence of the poisoned transitions, as designed: they do not reproduce the attacker's action, and their attacker-compliance rate under the trigger is 0.191 to 0.261 against the poisoned agents' 0.949 to 0.979. It does not follow that the agent is no longer exposed on those inputs. Its trigger invariance is 0.789 to 0.885, roughly twice that of the repaired agent, so the trigger still changes what it does, and under the same presentation protocol the retrained TD3+BC agent falls in 545 of 1,500 episodes against 35 for

TABLE V

ATTACKER CONTROL OVER THE THREE AGENTS. *Compliance* IS THE FRACTION OF TRIGGERED STATES ON WHICH THE POLICY'S ACTION IS CLOSER TO THE ATTACKER'S THAN TO THE CORRECTIVE ONE; *invariance* IS $\|\pi(s^+) - \pi(s^-)\|$. MEANS OVER THREE SEEDS, WITH STANDARD DEVIATION ON THE RETURN ROWS AND OMITTED ELSEWHERE, WHERE IT STAYS BELOW 0.05; THE LAST TWO ROWS ARE UNDER THE REAL TRIGGER, OUT OF 1,500 EPISODES.

		TD3+BC			**CQL**			**IQL**		
		Pois.	Rep.	Retr.	Pois.	Rep.	Retr.	Pois.	Rep.	Retr.
edit	compliance	0.989	**0.004**	0.254	0.958	**0.001**	0.198	0.973	**0.002**	0.190
	invariance	1.565	**0.295**	0.903	1.679	**0.384**	0.886	1.483	**0.356**	0.826
held-out	compliance	0.984	**0.102**	0.245	0.942	**0.097**	0.183	0.955	**0.089**	0.169
	invariance	1.558	**0.400**	0.873	1.665	**0.476**	0.826	1.452	**0.430**	0.780
unseen	compliance	0.979	**0.099**	0.261	0.949	**0.109**	0.198	0.959	**0.103**	0.191
	invariance	1.567	**0.417**	0.885	1.678	**0.502**	0.832	1.494	**0.448**	0.789
untriggered	$\|\pi - a_{\text{safe}}\|$	0.365	0.375	0.358	0.387	0.413	0.379	0.400	0.433	0.385
nominal return		3629 ± 8	3615 ± 14	3626 ± 6	3622 ± 19	3603 ± 25	3626 ± 7	3210 ± 482	3072 ± 726	3612 ± 55
in-env.	falls	725	**35**	545	543	**57**	425	1044	536	933
	return	2830 ± 203	$\mathbf{3591 \pm 28}$	2969 ± 439	3048 ± 320	$\mathbf{3571 \pm 50}$	3290 ± 150	2310 ± 178	2936 ± 875	2525 ± 577

the repaired one. Retraining removes data without specifying behavior: nothing in its training tells the agent what to do when those three velocity channels carry those particular values, and what it does there is whatever generalization produces, which here is measurably worse than a specified behavior.

Fine-tuning the deployed agent on the transitions known to be clean, with the 1,000 corrective actions reinserted in place, reaches an attacker compliance of 0.337 on unseen states and an invariance of 0.689 at 175k gradient steps, or 18% of a full training run, with no cost in preservation. The result is for TD3+BC at a single seed, and is the best the baseline reaches without losing nominal return; more gradient steps would likely lower it further.

The two remedies fail differently. Retraining reaches a lower compliance than fine-tuning, 0.261 against 0.337, but a higher invariance, 0.885 against 0.689: a policy retrained from scratch obeys the attacker less often while remaining more sensitive to the trigger, whereas one moved from the compromised weights retains a structure in which the trigger matters less while obeying it slightly more. Neither closes the channel, and PERU is below both on both quantities.

D. Localization of the Edit

A policy immune to the trigger might have installed a behavior at that point in observation space, or might have learned to discount the three dimensions altogether. Both predict immunity to the trigger and differ everywhere else, and a control separates them: the same three dimensions are overwritten with values drawn from a random row of the dataset, so that the triple is physically plausible and simply wrong for the current state. The last two rows of Table V report behavior under the real trigger, pooled over five presentation schedules, three at fixed intervals of 20, 40 and 80 steps and two randomized at the same expected rates, over 500 episodes per seed.

Under the real trigger the repaired agent falls in 35 of 1,500 episodes on TD3+BC and 57 on CQL; under the sham it falls in 758 and 497. The edit is therefore localized to the trigger point rather than being a general desensitization of those channels. All agents degrade comparably under the sham, at 926, 758 and 688 falls on TD3+BC. The degradation observed under the real trigger in the poisoned and retrained agents is therefore generic sensory corruption, and only the repaired agent's immunity is attributable to the edit. On IQL the same comparison gives 536 falls under the real trigger against 920 under the sham, a much narrower gap than on the other two algorithms. The poisoned IQL agents already fall in up to 47 of 100 episodes with no trigger at all, so in-environment measurements there are dominated by the instability of the starting agent rather than by the edit; the action-space measurements are unaffected and carry the smallest spread of the three.

Appendix F uses the same control to ask whether the backdoor responds only to the exact trigger.

VIII. DISCUSSION

A. Policy Editing Trade-offs

The retain loss is a soft constraint, not a guarantee. Network weights are shared across inputs, so changing the policy's behavior on the forget states changes weights that also determine outputs elsewhere. This is not specific to PERU: it is the difficulty continual learning addresses, where updating a network on new tasks risks degrading performance on previous ones [34]. Weight-space regularization in the style of Elastic Weight Consolidation [35] penalizes changes to parameters deemed important for the retain set, but in this setting it overconstrains the update: forget and retain states rely on shared internal representations, so preventing weight change prevents the edit. PERU constrains outputs instead of weights, which is what makes the edit feasible at all.

For the same reason, a convergence training phase of the kind TrajDeleter applies after unlearning is not compatible with our objective. Beyond the practical constraint that it requires a replay buffer, which two of our three access regimes do not have, its purpose conflicts with ours: TrajDeleter imposes no requirement on what should replace the forgotten behavior, so re-optimizing on the remaining data cannot

contradict its goal. PERU installs a specified behavior, and a phase that fine-tunes on retain data would overwrite it.

The gap between behavioral preservation and cumulative reward is wider in environments with sensitive dynamics, where a minor deviation early in an episode alters everything after it. Preservation is also only as informative as the retain set is representative: a set that fails to cover the states the policy actually visits can yield a high score while performance degrades in the regions it left out.

B. Retain Set Construction

When a replay buffer is available it can serve as the retain set, but roll-outs of π_{old} yield better post-edit reward. An offline dataset is confined to the state distribution of the behavior policy that collected it, which need not cover the regions the deployed policy visits, whereas roll-outs sample from its own occupancy measure, under which preservation is defined. The gap is widest on the medium-expert datasets, where the behavior policy is a mixture and the trained agent follows one mode of it. The optional random-action component during roll-out widens the coverage further, at the cost of retain states the policy visits only rarely.

C. Limitations

1) Scope of the membership claim: The claim made here is not that no auditor could determine whether a given trajectory was used in training, but that at the operating point of interest the question is not answered by the auditors considered; Section VI-D reports the measurement that supports this, together with the memorization gap that explains it. The claim is delimited in two directions. It concerns forget sets of a few trajectories drawn from the policy's own occupancy measure, in a regime where the model does not memorize them. Whether influence becomes detectable for forget sets that are much larger, or drawn from outside that measure, we do not determine. And it is a statement about achievable advantage rather than about the parameter distributions themselves: the auditors provide empirical upper bounds on what can be extracted, not an estimate of the distance between the distributions induced by the two training sets. Constructing trajectory-level auditors with adequate power in reinforcement learning remains an open problem.

2) Algorithm-dependent effectiveness: The effectiveness of the edit varies across algorithms in a way we can characterize but not explain. Six of the twenty-four configurations of Table IV fall well below the rest, and four of them use stochastic-policy algorithms, which is consistent with policies whose action distributions are broader and therefore harder to move toward a point target; we have no evidence isolating that mechanism. Until it is understood, the target fidelity attainable on a new algorithm cannot be predicted from the values reported here.

3) Critic-level information: For actor–critic algorithms PERU edits the actor and leaves the critic untouched, in line with the threat model: the critic plays no role at inference time and is often discarded after training, so a change confined to it is not a change in the deployed artifact. In deployments that do query the critic at runtime, for instance for online safety verification, residual information there could in principle be recovered by an adversary with white-box access. Extending PERU to edit both networks is straightforward and left to future work. For value-based methods such as DQN the distinction does not arise, since the same Q-network determines the deployed action; PERU uses a temperature-scaled softmax over its Q-values only as a differentiable editing representation.

4) High-dimensional observations: The synthetic retain set construction of Section V-A relies on a task-specific heuristic for pixel observations, demonstrated only on Pong. Pixel spaces are high-dimensional and their set of semantically valid frames is small and non-convex, so naive perturbations produce inputs the encoder was never trained on, and the resulting policy outputs are useless as retain targets. Unlike low-dimensional MuJoCo observations, where each dimension has a physical meaning and a known empirical range, pixel dimensions are not individually meaningful and their joint distribution is not characterized by simple bounds. A general environment-free retain generator for arbitrary pixel domains remains an open problem.

5) Per-task hyperparameter tuning: The main results rely on hyperparameters tuned per task–algorithm combination. The search is performed once and amortized over subsequent requests on the same agent, but it remains a setup cost on a new task. Configurations transfer reasonably well within an algorithm family, which reduces the search space in practice; automating that transfer, or removing the search through adaptive loss weights, is left to future work.

6) Scope of the case study: The case study covers one attack on one task, with the trigger BAFFLE specifies rather than one optimized for stealth; heuristic triggers of this form have been measured as suboptimal precisely on the criterion of preserving normal task performance [36]. Three seeds bound run-to-run variation but not variation over which transitions the attacker chooses to poison, which we fixed once so that all agents are audited on the same states, and the fine-tuning baseline uses a single seed.

IX. Conclusion

We presented PERU, a policy editing framework that treats reinforcement unlearning as behavior replacement: the data owner specifies what the agent should do on the affected states, while an auditor tests divergence from the recorded behavior. Across Atari and MuJoCo, on seven online and offline algorithms, PERU reaches a divergence of 99.0% from the recorded actions on forget trajectories while leaving it at 2.2% on retain trajectories, and repairs an agent compromised by a published poisoning attack from corrective actions covering under one percent of the affected transitions. Several directions remain open: generating retain states for high-dimensional observations without environment access, automating hyperparameter transfer within algorithm families, and extending the edit to the critic for deployments that query it at runtime.

ACKNOWLEDGEMENTS

This work was supported in part by NSF under Grants CNS-2312875 and OAC-2530896, DARPA under Cooperative Agreement D25AC00374-00, ONR under Award N00014-23-1-2221, and AFOSR under Award FA9550-23-1-0261.

ETHICS CONSIDERATIONS

This work addresses the problem of modifying deployed reinforcement learning agents when behaviors associated with previously collected trajectories must be replaced, for example following privacy-related deletion requests or the discovery of poisoned or unsafe training data. PERU operates at the behavioral level: it replaces the policy outputs associated with the affected states while preserving behavior elsewhere. It does not provide certified removal of a trajectory's training influence or, by itself, establish compliance with privacy regulations. Beyond privacy-motivated requests, the same mechanism can be used to repair unsafe behaviors discovered after deployment when a trusted process provides the desired replacement behavior.

The same capability could be misused by an adversary with white-box access to alter desirable or safety-critical behaviors. Our threat model therefore assumes that the data owner initiating the edit is trusted; preventing unauthorized model modification is a deployment-side responsibility outside the scope of PERU. The behavioral audit provides policy-level accountability by verifying whether the declared behavioral change occurred, but does not certify removal of training-data influence.

REFERENCES

[1] A. Mantelero, "The eu proposal for a general data protection regulation and the roots of the 'right to be forgotten'," *Computer Law & Security Review*, vol. 29, no. 3, pp. 229–235, 2013.

[2] S. L. Pardau, "The california consumer privacy act: Towards a european-style privacy regime in the united states," *J. Tech. L. & Pol'y*, vol. 23, p. 68, 2018.

[3] H. Zhang, T. Nakamura, T. Isohara, and K. Sakurai, "A review on machine unlearning," *SN Computer Science*, vol. 4, no. 4, p. 337, 2023.

[4] J. Xu, Z. Wu, C. Wang, and X. Jia, "Machine unlearning: Solutions and challenges," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 8, no. 3, pp. 2150–2168, 2024.

[5] D. Ye, T. Zhu, C. Zhu, D. Wang, K. Gao, Z. Shi, S. Shen, W. Zhou, and M. Xue, "Reinforcement unlearning," *arXiv preprint arXiv:2312.15910*, 2023.

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

[7] Y. Cao and J. Yang, "Towards making systems forget with machine unlearning," in *2015 IEEE symposium on security and privacy*. IEEE, 2015, pp. 463–480.

[8] A. Ginart, M. Guan, G. Valiant, and J. Y. Zou, "Making ai forget you: Data deletion in machine learning," *Advances in neural information processing systems*, vol. 32, 2019.

[9] E. Ullah, T. Mai, A. Rao, R. A. Rossi, and R. Arora, "Machine unlearning via algorithmic stability," in *Conference on Learning Theory*. PMLR, 2021, pp. 4126–4142.

[10] C. Guo, T. Goldstein, A. Hannun, and L. Van Der Maaten, "Certified data removal from machine learning models," *arXiv preprint arXiv:1911.03030*, 2019.

[11] L. Bourtoule, V. Chandrasekaran, C. A. Choquette-Choo, H. Jia, A. Travers, B. Zhang, D. Lie, and N. Papernot, "Machine unlearning," in *2021 IEEE symposium on security and privacy (SP)*. IEEE, 2021, pp. 141–159.

[12] A. Golatkar, A. Achille, and S. Soatto, "Eternal sunshine of the spotless net: Selective forgetting in deep networks," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 9304–9312.

[13] 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)*. IEEE, 2017, pp. 3–18.

[14] N. Carlini, S. Chien, M. Nasr, S. Song, A. Terzis, and F. Tramer, "Membership inference attacks from first principles," in *2022 IEEE symposium on security and privacy (SP)*. IEEE, 2022, pp. 1897–1914.

[15] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha, "Privacy risk in machine learning: Analyzing the connection to overfitting," in *2018 IEEE 31st computer security foundations symposium (CSF)*. IEEE, 2018, pp. 268–282.

[16] P. W. Koh and P. Liang, "Understanding black-box predictions via influence functions," in *International conference on machine learning*. Pmlr, 2017, pp. 1885–1894.

[17] M. van Otterlo and M. Wiering, *Reinforcement Learning and Markov Decision Processes*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 3–42.

[18] R. S. Sutton, A. G. Barto *et al.*, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[19] R. M. Nkhumise, D. Basu, T. J. Prescott, and A. Gilra, "How does your rl agent explore? an optimal transport analysis of occupancy measure trajectories," *arXiv preprint arXiv:2402.09113*, 2024.

[20] S. Kullback and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79–86, 1951.

[21] F. E. Grubbs, "Procedures for detecting outlying observations in samples," *Technometrics*, vol. 11, no. 1, pp. 1–21, 1969.

[22] 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.

[23] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of artificial intelligence research*, vol. 47, pp. 253–279, 2013.

[24] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

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

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[27] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[28] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. Pmlr, 2018, pp. 1861–1870.

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

[30] A. Kumar, A. Zhou, G. Tucker, and S. Levine, "Conservative q-learning for offline reinforcement learning," *Advances in neural information processing systems*, vol. 33, pp. 1179–1191, 2020.

[31] S. Fujimoto and S. S. Gu, "A minimalist approach to offline reinforcement learning," *Advances in neural information processing systems*, vol. 34, pp. 20 132–20 145, 2021.

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

[33] C. Gong, Z. Yang, Y. Bai, J. He, J. Shi, K. Li, A. Sinha, B. Xu, X. Hou, D. Lo *et al.*, "Baffle: Hiding backdoors in offline reinforcement learning datasets," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 2086–2104.

[34] G. I. Parisi, R. Kemker, J. L. Part, C. Kanan, and S. Wermter, "Continual lifelong learning with neural networks: A review," *Neural networks*, vol. 113, pp. 54–71, 2019.

[35] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Pro-*

ceedings of the national academy of sciences, vol. 114, no. 13, pp. 3521–3526, 2017.
[36] M. Zhang, O. Ma, K. Wei, S. Li, and S. Ji, "Toobadrl: Trigger optimization to boost effectiveness of backdoor attacks on deep reinforcement learning," *arXiv preprint arXiv:2506.09562*, 2025.
[37] S. Kumar, "Balancing a cartpole system with reinforcement learning–a tutorial," *arXiv preprint arXiv:2006.04938*, 2020.
[38] R. S. Sadavarte, R. Raj, and B. S. Babu, "Solving the lunar lander problem using reinforcement learning," in *2021 IEEE International Conference on Computation System and Information Technology for Sustainable Solutions (CSITSS)*. IEEE, 2021, pp. 1–6.
[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.
[40] N. Agarwal, B. Bullins, and E. Hazan, "Second-order stochastic optimization for machine learning in linear time," *Journal of Machine Learning Research*, vol. 18, no. 116, pp. 1–40, 2017.
[41] K. P. Murphy, *Probabilistic machine learning: Advanced topics*. MIT press, 2023.

APPENDIX

A. Experimental Setup

1) Observation and Action Spaces: The shape of the policy, and therefore what an edit has to modify, depends on the topology of the observation and action spaces. We distinguish three setups. *Discrete control with low-dimensional observations* has $\mathcal{O} = \mathbb{R}^{d_o}$ with $d_o \ll 100$ and a finite action set, as in CartPole [37] or LunarLander [38], and admits value-based methods or categorical policy-gradient variants. *Continuous control with low-dimensional observations* covers the MuJoCo locomotion tasks, where the agent receives joint positions and velocities and outputs a real vector of torques, and is addressed by actor–critic algorithms with deterministic or Gaussian policies. *Image-based domains* such as Atari combine large observation spaces with discrete actions, and are treated with the same categorical objectives as the first setup. The environments below cover the second and third; the first is a special case of the third in everything that concerns the edit.

2) Environments: For discrete action spaces we use three environments from the Atari 2600 suite through the Arcade Learning Environment [23]: `PongNoFrameskip-v4`, `EnduroNoFrameskip-v4` and `SpaceInvadersNoFrameskip-v4`. Observations are stacked into four grayscale frames, which supplies the agent with temporal information such as object velocity and direction. This affects the structure of the feature extractor, typically a convolutional network, but not the editing procedure. For continuous control we use three MuJoCo locomotion environments [22]: Walker2d, Hopper and HalfCheetah, with observation dimensionalities between 11 and 17 and actions in $(-1, 1)^{d_a}$.

3) Algorithms: On Atari we use DQN [25], [39], a value-based method approximating Q-functions with convolutional networks and ϵ-greedy action selection, and PPO [26], a policy-gradient method learning stochastic policies through clipped surrogate objectives with shared convolutional encoders. For continuous control we use three online algorithms: TD3 [27], an off-policy actor–critic with deterministic policies; SAC [28], which maintains stochastic Gaussian policies with state-dependent mean and variance; and PPO adapted to continuous control, outputting diagonal Gaussian action distributions. Among offline methods trained on D4RL datasets [32] we use IQL [29], which learns an implicit policy through expectile regression; CQL [30], which adds a conservative penalty to Q-values; and TD3+BC [31], which combines TD3 with a behavioral cloning regularizer.

4) Hardware and Software: All the experiments were conducted on a server with an NVIDIA A100 80GB GPU and 500GB of RAM. Energy measurements were conducted on a Jetson Orin Nano with 8GB of RAM, with power consumption monitored at 1 Hz through the `jtop` library. For online algorithms we used Stable Baselines3 v2.6.0, with pretrained models from the Stable Baselines3 Zoo repository; for offline algorithms, d3rlpy v2.6.0 with models trained on the D4RL medium-expert datasets. The editing procedure is implemented in PyTorch v2.7.0 and is modular, so that it applies to any library exposing access to policy networks.

B. Hyperparameter Search

We performed a grid search to identify a configuration for each algorithm–task combination, using forget trajectories of 500 states. The search space included the loss weights λ_f and λ_r; the retain set size $|D_r|$; the minimum distance threshold used to filter retain states too close to the forget set; the probability of a random action during environment roll-out; the learning rate; and the maximum gradient norm for clipping. Two parameters are algorithm-specific: the temperatures shaping the action distributions in continuous control, where deterministic policies allow separate values for the current and the target distribution, and the softmax temperature with its Q-value scaling factor in discrete control. For each combination we selected the configuration giving the best trade-off among target fidelity, behavioral preservation and cumulative reward.

The per-task tuning is performed once and reused across unlearning requests, which suits the operational scenarios of Section IV, where unlearning is a rare event rather than a continuous workload. Appendix D reports the sensitivity of the method to the three parameters that matter most. The selected values for every algorithm–task combination are released with the code.

C. Influence-Based Auditors: Estimation Details

1) Experimental Design: For each of IQL, CQL and TD3+BC we train five policies on the full dataset and five on the dataset with the ten forget episodes removed, using paired seeds so that the two models of each pair share initialization and hyperparameters. Any scaler fitted from data is frozen to the statistics of the full dataset and passed identically to both arms: a dataset that differs by ten episodes induces a slightly different normalization, and all four statistics are gradient-based and would be shifted by it. The largest shift avoided this way is 0.002 standard deviations, so the confound is small, but it acts on forget and retain states alike and would only add noise.

Two further choices remove confounds that would otherwise be mistaken for signal. All audited trajectories contribute the same fixed window of 450 states, because gradient-based statistics scale with the number of states. And forget and retain trajectories are drawn from the same pool of episodes long enough to provide that window, so that trajectory length is not a proxy for membership.

2) Normalization and Control: The eighty retain trajectories are split into two disjoint halves: one provides the per-model median used to normalize scale differences across models, while the other serves as the control. Using the same trajectories for both purposes would make the normalized control distribution self-referential: its median is one in every model by construction, artificially driving the control ROC-AUC toward 0.5. In simulation, the AUC remains 0.503 even when scores in one model group are uniformly increased by 30%. With disjoint halves, control and forget trajectories undergo the same normalization procedure without this dependence, making their statistics comparable; accordingly, the null band widens from ± 0.027 to ± 0.097, which we use for both columns.

3) The Four Statistics: All four statistics are computed with respect to a behavior-cloning surrogate for the training loss, namely the squared error between the policy's action and the recorded one. For TD3+BC this surrogate corresponds to the behavior-cloning term of the actor objective; for CQL and IQL it is only an approximation.

At zeroth order, we use the surrogate loss on the audited trajectory, the canonical membership statistic [15]: training samples are expected to have lower loss. At first order, we use the gradient norm on the audited state–action pairs and its cosine alignment with the mean gradient over a held-out set of retain trajectories disjoint from the control set. At second order, we use curvature-aware self-influence [16], evaluated as $g^\top (H + \lambda I)^{-1} g$.

The inverse-Hessian-vector product is obtained by conjugate gradient on the damped system rather than by the truncated Neumann series of [40]. Because the parameters were obtained by optimizing a different objective, they are not necessarily at a minimum of the surrogate loss; in practice, we observe negative Hessian eigenvalues, for which the Neumann recursion is unstable without sufficient damping. We estimate the spectral norm by power iteration and initialize the damping above this estimate, doubling it whenever conjugate gradient encounters negative curvature. The relative residual is below 10^{-4} in all reported runs.

The required damping is of the same order as the estimated spectral norm, substantially reducing the contribution of curvature. Consequently, self-influence approaches a scaled squared gradient norm as damping increases. Empirically, their ROC-AUCs differ by at most 0.011 in every configuration of Table III. Thus, among the four statistics spanning three orders of the signal, second-order information provides no measurable advantage over the first-order gradient norm in this regime.

4) Additional Statistics: Besides the pooled area under the curve reported in Table III, we compute a per-trajectory verdict by comparing, for each audited trajectory, the five values from the with-forget policies against the five from the without-forget policies. With five models per arm, the probability that a single trajectory reaches an area under the curve of 1.0 by chance is $1/\binom{10}{5} = 0.004$, so among ten trajectories 0.04 are expected to do so; none did in any configuration. We also compute a parametric variant in the style of LiRA, reporting the separation d' between the two arms; because d' contains an absolute value it is positive even under the null, so we report the same quantity on the control trajectories as a noise floor, and it is comparable to or larger than the forget-side value in most configurations. Finally, the true-positive rate at a false-positive rate of 10% does not exceed 0.24 in any configuration.

5) Scope: An auditor that additionally knows the target φ can identify which states were edited. This reveals the behavioral change the data owner declared, not residual training influence, and we do not count it as membership inference.

D. Ablations

1) Forget-Side Divergence and the Retain Term: Table VI compares four choices for the forget and retain terms on SAC-Walker2D when unlearning one trajectory, holding λ_f, λ_r and the retain set fixed. Forward KL gives the best trade-off, with a target fidelity of 80.9% at 91.1% behavioral preservation and a cumulative reward of 2878.8. Squared error in action space preserves comparably, 92.2%, but reaches 12 points less target fidelity, 68.5%. Reverse KL and Jensen-Shannon produce almost no edit, 2.9% and 5.3%, even though they preserve behavior and reward at levels at or above forward KL.

The difference follows from the zero-forcing property of forward KL [41]. When the target assigns near-zero probability to the forget action, forward KL penalizes any residual mass the policy places there, and the gradient pushing it down is large. Reverse KL weights each action by its probability under the target rather than under the policy, so it gives almost no weight to the forget action and almost no signal to remove it. Jensen-Shannon is bounded above and cannot produce a comparable gradient even when the two distributions disagree sharply.

The same table answers whether the retain term is needed at all, given that the forget term alone already drives the policy away from the original behavior. It is: with $\lambda_r = 0$, target fidelity reaches 99.4% while behavioral preservation drops to zero and cumulative reward becomes negative, -4.3. Optimizing the forget term alone destroys the policy everywhere, not only on the forget states.

2) Hyperparameter Sensitivity: We sweep a $4 \times 4 \times 4$ grid over the retain weight λ_r, the retain set size $|D_r|$ and the minimum distance threshold between forget and retain states, on TD3 across the three MuJoCo environments, for 192 configurations in total. Each is evaluated with three independent unlearning runs using different forget trajectories of 500

states, and the resulting policy is evaluated over 100 episodes. Figure 4 reports means with 95% confidence intervals.

The retain weight, with λ_f fixed at one, has an environment-dependent effect on target fidelity: Hopper is insensitive, staying near 86–87%; HalfCheetah declines as λ_r grows; Walker2d improves from 88% to 97%. Behavioral preservation stays above 96% throughout, which indicates that the retain objective is already satisfied at low λ_r, and cumulative reward rises mildly with λ_r in all three environments.

The retain set size matters most on HalfCheetah, where target fidelity is non-monotonic with a drop at $|D_r| = 200$ that coincides with a drop in reward, suggesting that intermediate sizes can fail to capture the policy's behavior. Walker2d is stable across sizes. All environments improve in behavioral preservation as the set grows.

The distance threshold has its strongest effect on Hopper, where target fidelity improves from 77% to 87% as the threshold goes from 0.1 to 0.5 and then plateaus: small thresholds admit retain states too similar to the forget states, which works against the edit. Rewards are stable across most values, with the exception of Hopper at a threshold of 1.5, where the confidence interval widens considerably, consistent with excessive filtering leaving a sparse retain set. HalfCheetah peaks at 1.0 and Walker2d is insensitive.

E. Target Construction

When no replace dataset is supplied, φ is built from the original policy. The construction differs by action-space type.

For discrete actions, policy-gradient methods such as PPO give action probabilities directly, while for value-based methods such as DQN we convert Q-values into a distribution through a softmax with temperature τ:

$$\pi_\theta(a|s) = \frac{\exp(Q_\theta(s,a)/\tau)}{\sum_{a'} \exp(Q_\theta(s,a')/\tau)}.$$

We then set $\varphi(a_f|s) \approx 0$ on the forget action and redistribute the removed mass among the remaining $|\mathcal{A}| - 1$ actions, by one of two rules. The *ranking-preserving* rule redistributes the forgotten action's probability mass equally among the remaining actions, preserving their original ranking: $\varphi(a_k|s) = \pi_{old}(a_k|s) + \pi_{old}(a_f|s)/(|\mathcal{A}|-1)$ for $k \neq f$; the *uniform* rule assigns equal probability to every non-forgotten action, making the agent agnostic on the forget state. Assigning exactly zero to a_f would make the KL infinite whenever π_{old} places any mass there, so we clamp to a small $\epsilon > 0$.

For continuous actions the target is a Gaussian $\mathcal{N}(\mu_{\text{tar}}, \sigma^2_{\text{tar}} I)$, and two rules set its mean. The *opposite-torque* rule sets $\mu_{\text{tar}}(s) = -\mu_{old}(s)$, which guarantees a Euclidean distance of $2\|\mu_{old}\|$ from the original action and therefore produces strong gradients. The *random-shift* rule draws a mean uniformly within the action range, producing diverse targets without enforcing a minimum distance, which may yield weaker gradients. When a replace dataset provides explicit actions, φ is centered on those with the assigned variance.

Figure 5 compares the two pairs of rules, on Walker2D for continuous control and SpaceInvaders for discrete control. Each unlearning run forgets one complete episode, and we report means over ten runs with different randomly sampled trajectories. For continuous control, opposite-torque reaches substantially higher target fidelity than random-shift on every algorithm: 35.7% against 7.2% on PPO, 80.9% against 59.1% on SAC, and 91.8% against 49.9% on TD3. The gap follows from the guaranteed distance, whereas a random target often falls close to the original action. Behavioral preservation is high for both, between 91% and 98%. For discrete control the two rules behave similarly: DQN reaches 88.6% with the ranking-preserving rule against 81.9% with the uniform one, while PPO shows the opposite ordering, 81.9% against 91.0%, with behavioral preservation near 99% in all cases. We adopt opposite-torque for continuous control and the ranking-preserving rule for discrete control as defaults.

F. Attack Surface under the Sham Trigger

The localization control of Section VII-D also measures how wide the attack surface is. Under the sham trigger the poisoned agent's attacker-compliance rate is 0.465–0.514 across the three algorithms, close to the 0.5 a policy indifferent between the two reference actions would give: the backdoor partially fires on velocity readings near the trigger but not equal to it, so the surface is wider than the attacker specified. The repaired agent is at 0.258–0.291 and the retrained one at 0.216–0.296, with no systematic ordering between them, so the edit removes that partial response as well and not only the response to the exact trigger.

G. Time and Energy

We compare `PERU` against retraining from scratch on a Jetson Orin Nano with 8 GB of RAM, chosen as representative of the constrained hardware on which a deployed agent might have to be repaired in place. We measure wall-clock time and integrate instantaneous power consumption, sampled at 1 Hz through the `jtop` library, over the entire procedure.

The retraining baseline here differs from those of Section VI-C: rather than removing the forget trajectories or injecting a target, we train a new TD3 agent with Stable Baselines3, using the default hyperparameters from Stable Baselines Zoo, and apply a negative reward penalty whenever the agent visits a state $s \in S_f$ and selects an action close

TABLE VI
ABLATION OF THE PERU LOSS DESIGN ON SAC-WALKER2D WHEN UNLEARNING ONE TRAJECTORY (MEAN ± 95% CI OVER TEN RUNS). WE COMPARE ALTERNATIVE FORGET LOSSES AND THE REMOVAL OF THE RETAIN TERM; λ_f, λ_r AND THE RETAIN SET ARE OTHERWISE FIXED.

Variant	TF ↑	BP ↑	Reward ↑
Forward KL (ours)	$\mathbf{80.9 \pm 1.3}$	$\mathbf{91.1 \pm 0.4}$	$\mathbf{2878.8 \pm 307.5}$
$\lambda_r = 0$	99.4 ± 0.1	0.0	-4.3 ± 4.3
MSE	68.5 ± 2.8	92.2 ± 0.3	3074.9 ± 345.6
Reverse KL	2.9 ± 2.5	96.0 ± 0.1	3677.3 ± 119.9
Jensen–Shannon	5.3 ± 5.8	97.5 ± 0.2	3700.5 ± 133.2

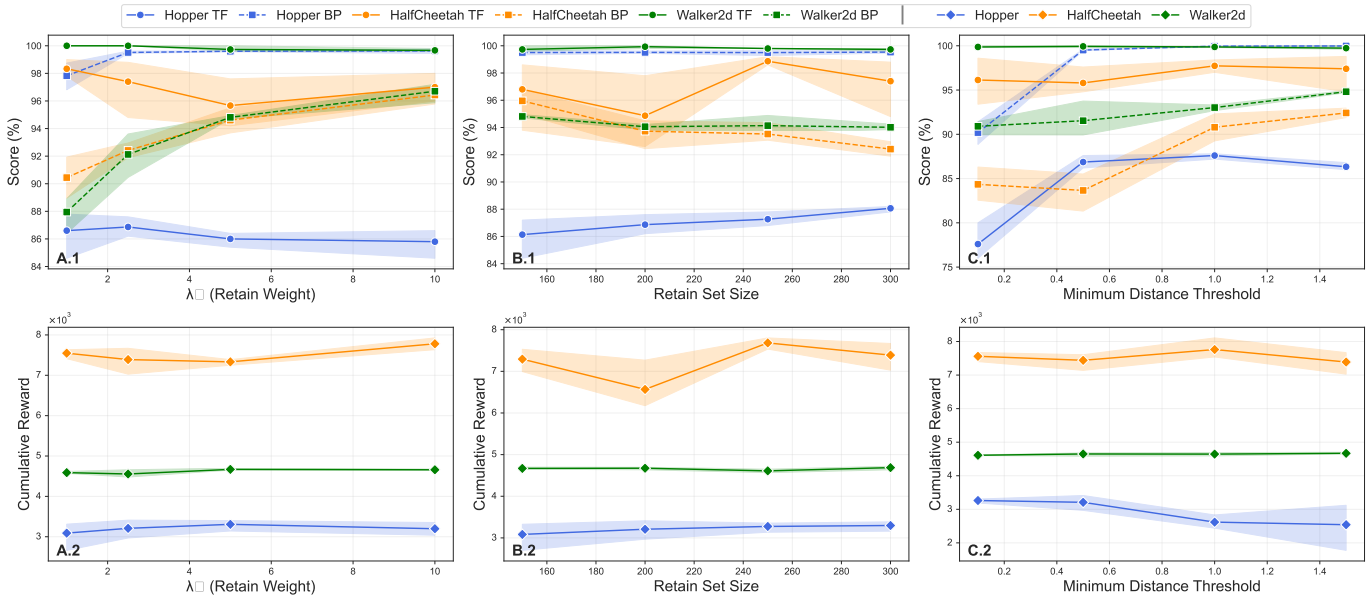

Fig. 4. Hyperparameter sensitivity of PERU on TD3 across MuJoCo environments. Top row: target fidelity and behavioral preservation. Bottom row: cumulative reward. Columns: retain weight λ_r, retain set size $|D_r|$, distance threshold. Shaded regions are 95% confidence intervals over three runs.

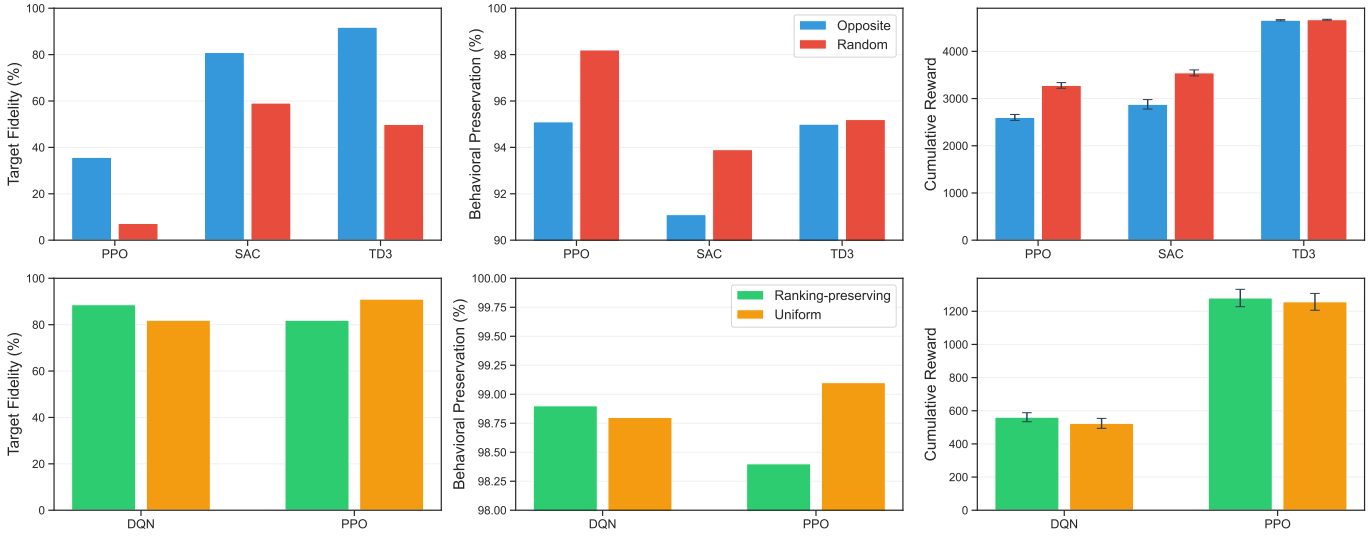

Fig. 5. Target construction rules: opposite-torque against random-shift for continuous control (Walker2D, top), and ranking-preserving against uniform redistribution for discrete control (SpaceInvaders, bottom).

to the corresponding forget action a_f. The penalty gives the new policy an explicit incentive to avoid the original behavior on those states, although it guarantees nothing: the agent may still visit them if they lie on a high-reward trajectory. We use this variant because its cost is the one directly comparable to a full online training run.

Table VII reports the comparison on the three MuJoCo locomotion tasks when unlearning a single full-episode trajectory. PERU is up to 25× faster and uses up to 12× less energy, with the margin varying by task. Walker2D and Hopper show the largest gains; HalfCheetah requires approximately twice as many retain states to maintain post-edit performance, which reduces the relative saving. These margins make PERU compatible with deployment scenarios in which the repair must be executed on the device, whereas retraining from scratch would require either offloading to larger infrastructure or accepting the longer training time.

TABLE VII
WALL-CLOCK TIME AND ENERGY FOR RETRAINING FROM SCRATCH AND FOR PERU ON A JETSON ORIN NANO, WHEN UNLEARNING ONE FULL TRAJECTORY WITH TD3.

	Time (s)		**Energy (kJ)**	
Task	Retrain	PERU	Retrain	PERU
Walker2D	33,900	1,380	166.0	13.3
Hopper	33,300	1,368	163.0	13.2
HalfCheetah	33,009	3,820	163.0	37.4