Continual Preference-Based Reinforcement Learning

Videh Raj Nema **Sooraj Sathish** **Calarina Muslimani** **Srijita Das**
Matthew E. Taylor

Keywords: Preference-based Reinforcement Learning, Continual Learning, Hypernetworks

Summary

In reinforcement learning (RL), specifying reward functions accurately aligned with user preferences can be difficult. A potential solution is preference-based RL that learns a policy and a reward function from preferences over behaviors given by a teacher (e.g., a human and/or an AI system). Most existing works assume the teacher's model of preferences (underlying reward function) is static. However, real-world interactions can often be non-stationary, where the agent is expected to solve multiple sequential tasks characterized by the changing teacher preferences. Hence, we propose a continual preference-based RL setting involving continual learning of reward functions, where the objective is to balance learning new and preserving old knowledge. For this, we propose using task-conditioned regularized hypernetworks that achieve both objectives and learn aligned reward functions. We evaluate our method with simulated human feedback on five simulated robotic manipulation and locomotion domains and show that it performs better than (or competitive to) several continual learning baselines.

Contribution(s)

1. We formalize *Continual Preference-Based Reinforcement Learning* (CPRL), a setting involving continual learning of reward functions from a sequence of non-stationary teacher preferences without access to past preference data. CPRL bridges the gap between conventional preference-based RL and continual learning.
   Context: Majority existing preference-based RL methods assume that preferences are stationary and/or that there is only one task (Wirth et al., 2017; Lee et al., 2021a). CPRL explicitly considers the continual learning context, where multiple reward functions (indicating teacher preferences) must be learned. The objective is to simultaneously adapt to new tasks (reward functions) and remember how to solve previously learned tasks.
2. We construct a hypernetwork-based algorithm with task-specific conditioning for continual reward learning in preference-based RL. The hypernetwork generates task-specific reward parameters conditioned on the self-learned task-specific contexts, enabling a single shared meta-model to represent multiple reward functions across tasks. We apply an output-space regularization mechanism that mitigates catastrophic forgetting by anchoring previously generated reward functions. We demonstrate the efficacy of our method in terms of the CPRL objectives and alignment with the teacher's preferences on five simulated robotic manipulation and locomotion domains.
   Context: Hypernetworks have been previously explored in supervised and model-based RL settings. We extend the continual learning approach based on task-conditioned hypernetworks by Von Oswald et al. (2019) to overcome the limitations of the standard preference-based RL setting. Our setup is different and more challenging than theirs due to dynamic preference data collection and the potential for distributional shifts (Section 4). Unlike the conventional regularization-based methods (as categorized by Van de Ven et al. (2022)) that operate in the learned model's parameter space, the hypernetwork-based approach operates in the output space of the hypernetwork. Combined with the regularizer, this results in greater flexibility without needing to revisit and retrain on past preference data.

Continual Preference-Based Reinforcement Learning

Videh Raj Nema[1,3*] **Sooraj Sathish**[2*] **Calarina Muslimani**[3]
Srijita Das[4] **Matthew E. Taylor**[3]
[1]da25d003@smail.iitm.ac.in

[1]IIT Madras
[2]IIIT Bangalore
[3]University of Alberta
[4]University of Michigan–Dearborn
[*]*Co-first authors*: The author contributions are specified in Section 7.

Abstract

In reinforcement learning (RL), specifying reward functions accurately aligned with user preferences can be difficult. A potential solution is preference-based RL that learns a policy and a reward function from preferences over behaviors given by a teacher (e.g., a human and/or an AI system). Most existing works assume the teacher's model of preferences (underlying reward function) is static. However, real-world interactions can often be non-stationary, where the agent is expected to solve multiple sequential tasks characterized by the changing teacher preferences. Hence, we propose a continual preference-based RL setting involving continual learning of reward functions, where the objective is to balance learning new and preserving old knowledge. For this, we propose using task-conditioned regularized hypernetworks that achieve both objectives and learn aligned reward functions. We evaluate our method with simulated human feedback on five simulated robotic manipulation and locomotion domains and show that it performs better than (or competitive to) several continual learning baselines.

1 Introduction

Deep reinforcement learning has been successfully used to solve many complex control tasks (Mnih et al., 2015; Haarnoja et al., 2018). However, an RL agent's ability to solve a task is heavily dependent on the human-engineered reward function. Designing informative and dense rewards that accurately reflect the user's intentions is difficult and can led to undesirable behaviors (Amodei et al., 2016). The problem only becomes more prominent when the agent needs to solve multiple tasks and/or the reward function is non-stationary.

Preference-based RL (PbRL) (Christiano et al., 2017) has become an increasingly popular paradigm that leverages human preferences to learn a reward function. This approach eliminates manual reward function specification by using easier-to-provide human (teacher) preference comparisons. Majority PbRL methods (Lee et al., 2021a; Park et al., 2022) assume that the teacher's preferences are stationary and that there is only one task. However, real-world human-agent interactions can often be non-stationary, where the agent is expected to solve multiple sequential tasks having different objectives. Further, there may be more than one humans in the loop, i.e., humans with different preferences interacting with the agent at different points in time. For instance, a helper robot in a restaurant needs to cater to the preferences of different customers coming over time. Hence, in this work, we model the agent's interaction with the teacher as a continual learning process.

Continual learning (CL) refers to constant and incremental learning of new information and behaviors from a non-stationary stream of data (Van de Ven et al., 2022). Typically, only a small portion of

data is available at once. A simple approach to CL is fine-tuning, i.e., training the model incrementally on each task using only the loss on the current data. This approach generally suffers from the phenomenon of catastrophic forgetting (Rusu et al., 2016). We confirm the same in our experiments (Appendix 8). Although forgetting can be overcome to some extent via retraining on previous task data, this becomes infeasible due to the dramatic growth in the size of collected experience.

Addressing the above-stated limitations of learning from preferences, we introduce continual preference-based RL (CPRL). In this problem setting, the agent needs to continually learn newly encountered reward functions (given by non-stationary behavioral preferences of teachers) in a sequence of tasks using changing preferences. Further, the objectives in this setting are to (1) continually learn new knowledge (tasks) and (2) preserve old knowledge (avoid forgetting). These two objectives can be conflicting and the agent needs to balance both (Carpenter & Grossberg, 1987). As a solution method for CPRL, we propose extending the continual learning approach based on task-conditioned hypernetworks by Von Oswald et al. (2019) to overcome the limitations of the standard PbRL setting. We use fixed-capacity hypernetworks conditioned on a task-specific context vector to generate parameters of the reward function. The hypernetwork is trained on each new task such that its input-output mapping does not change substantially on previous tasks, thereby balancing the dual objectives. The training is done with a regularizer based on the hypernetwork outputs such that there is no need to revisit and retrain on previous tasks. This allows a single meta-model to represent multiple reward functions while sharing parameters across tasks. To summarize our contributions:

- We propose and formalize the CPRL setting, bridging the gap between PbRL & continual learning.
- We extend the hypernetwork-based approach to CPRL and show that they can continually learn new reward functions (tasks) that can be optimized to obtain desirable policies without considerably forgetting previous tasks (dual objectives).
- We evaluate our method on manipulation and locomotion tasks and show that it performs better than (or competitive to) various CL baselines regarding dual objectives and learns reward functions better aligned with teacher preferences. We also show encouraging results on forward transfer.

2 Background & Related Work

Preference-based Reinforcement Learning In PbRL, we alternate between learning a reward function that adheres to the teacher's preferences and learning a policy using that reward. The agent interacts with the environment to collect a set of trajectory segments (σ). It learns a reward function by querying the teacher for (typically binary) preferences over trajectory segments (σ^0, σ^1). The teacher compares the segments and provides a feedback label $y \in \{0, 1\}$, where $y = 0$ if the first segment is preferable over the second and $y = 1$ vice-versa. The probability of preferring σ^0 over σ^1 ($\sigma^0 \succ \sigma^1$) is modeled using Bradley & Terry (1952), as proposed by Christiano et al. (2017):

$$P_\psi[\sigma^0 \succ \sigma^1] = \frac{\exp \sum_{(s,a)\in\sigma^0} r_\psi(s,a)}{\exp \sum_{(s,a)\in\sigma^0} r_\psi(s,a) + \exp \sum_{(s,a)\in\sigma^1} r_\psi(s,a)}. \tag{1}$$

The reward parameters ψ are updated by minimizing the cross-entropy loss between the predicted preference probabilities and the teacher's provided labels y over a dataset of queries. The policy can be optimized using the learned reward using a deep RL algorithm. PbRL has recently gained traction due to its numerous benefits, such as avoiding reward engineering, reward exploitation, and ease of working with non-experts (Wirth et al., 2017; Lee et al., 2021b).

Hypernetworks for Continual Learning Hypernetworks $\mathcal{H}_\Theta$ are meta-models parameterized by Θ that generate the parameters ψ of another network called main network $\mathcal{F}_\psi$ (reward network in our case) based on context-specific information $\mathcal{C}$, i.e., $\psi = \mathcal{H}_\Theta(\mathcal{C})$ (Ha et al., 2017; Schmidhuber, 1992). The context $\mathcal{C}$ provides task-specific information to the hypernetwork, which is used to produce ψ. The main network acts as a placeholder and uses the generated parameters to compute predictions $\hat{y} = \mathcal{F}_\psi(x)$ and an associated loss $\mathcal{L}$ (for (x, y) datapoints). During the backward pass,

the parameters ψ are not directly updated. Instead, the gradients are backpropagated through $\mathcal{F}_\psi$ to update the hypernetwork parameters Θ (and optionally $\mathcal{C}$) via $\nabla_\Theta \mathcal{L}$ (and $\nabla_\mathcal{C} \mathcal{L}$).

Hypernetworks have been effective for a multitude of problems across vision, language, and RL due to their flexibility and adaptability (Chauhan et al., 2024). Relevant to our work, they have been shown to be useful in CL for classification and generative models (Von Oswald et al., 2019). In context of RL, Huang et al. (2021) use hypernetworks to continually learn the transition dynamics. Complementing our work in human-in-the-loop RL, Auddy et al. (2023) show that hypernetworks can learn robotic skills from demonstrations. Different from these works, we show that hypernetworks can effectively learn reward functions from non-stationary teacher preferences in CL.

Continual Learning from Preferences While CL has been explored in RL and supervised learning, its intersection with preference-based learning is nascent. Recent works like CPPO (Zhang et al., 2024) and COPR (Zhang et al., 2025) address continual alignment of Large Language Models (LLMs) from human feedback. CPPO uses a sample weighting strategy to balance policy learning and knowledge retention. COPR is an RL-free algorithm that uses past training data to regularize the current policy. Our work differs in some key aspects: (1) we focus on the continual learning of the reward function rather than the policy, (2) ours is a more general application-agnostic framework over RLHF for LLMs, and (3) our work is applicable to any RL algorithm than just PPO (as in CPPO). More recently, Pref-GUIDE addresses non-stationarity in real-time scalar feedback by converting it into temporally local preference pairs (Ji & Chen, 2025). However, it focuses on mitigating concept drift *within* a single task to enable continued training after feedback ceases. In contrast, CPRL targets a multi-task sequential setting where the objective is to learn and retain a sequence of distinct reward functions (tasks) without forgetting previous tasks.

3 The Continual Preference-Based RL Problem Setting

The CPRL setting involves continual reward learning using non-stationary behavioral preferences of human/artificial teacher(s). The agent faces several tasks sequentially, each differing in the (internal) preference model $\mathcal{M}$ of the teacher(s). Using these changing preferences, the agent must learn a (parameterized) reward function r_ψ, interleaving reward learning and policy optimization to capture task-specific preference models $\mathcal{M}^i$ (for task i). The overall objective is: (1) adapt to changing teacher preferences and (2) avoid forgetting how to solve previous tasks.

We assume that the agent (1) only has access to the current task's preferences and cannot revisit past preference data, and (2) it knows when a task switch occurs. We focus only on learning the reward function continually because it entails complete information about the changing preferences. The policy just follows the changed preferences encoded in the reward function. Hence, a new policy is initialized and learned for every new task. Additionally, reward learning (via hypernetworks) can be simpler than learning a policy directly because intuitively, it is easier to learn, remember, and transfer knowledge about what the task objective is than how to achieve that objective.

4 Methodology

Now, we describe the hypernetwork-based algorithm for CPRL. Hypernetworks have been explored in supervised CL by Von Oswald et al. (2019). In supervised CL, the agent has access to all data points (for a particular task) at once; however in CPRL, preference data is collected dynamically as the teacher provides labels. Further, the collection of new data points depends on the changing reward and policy, and this data is, in turn, again used to update the reward. This can cause distributional shifts, resulting in cases difficult to recover.

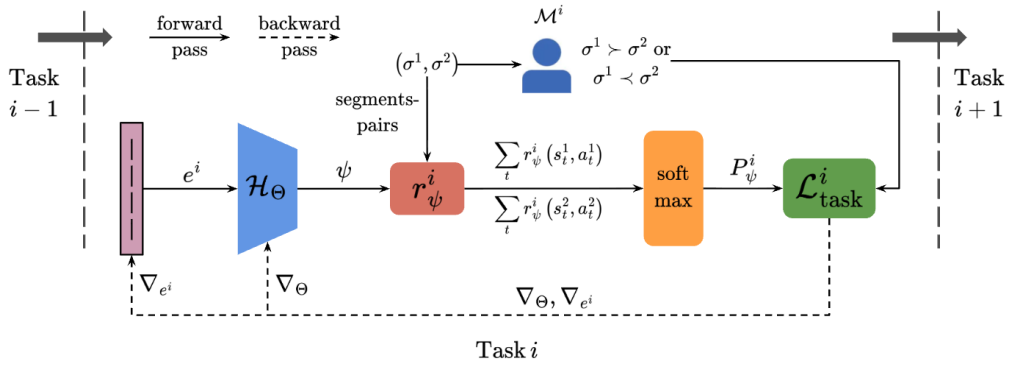


Figure 1: Reward learning with hypernetworks in CPRL for task i (teacher $\mathcal{M}^i$).

4.1 Hypernetworks for CPRL

We learn the parameters of a task-conditioned hypernetwork $\mathcal{H}_\Theta$ (Figure 1) that, in turn, generates the parameters, ψ of the reward network. $\mathcal{H}_\Theta$ is shared across all tasks, but the context (e_i) and the generated reward network (r^i_ψ) are specific to task i. The embedding e_i is a low-dimensional vector representing task i and is learned via backpropagation. The return for each trajectory segment is computed using r^i_ψ to compute the preference predictor P^i_ψ (Equation 1). We then use P^i_ψ and the labels $y \in \{0, 1\}$ from the teacher to construct the task-specific cross-entropy loss given by:

$$\mathcal{L}^i_{\text{task}}(\sigma^0, \sigma^1, y; \Theta, e^i) = -\mathbb{E}_{(\sigma^0,\sigma^1,y)\sim\mathcal{B}^i}\Big[(1-y)\log P^i_\psi[\sigma^0 \succ \sigma^1] + y \log P^i_\psi[\sigma^1 \succ \sigma^0]\Big], \quad (2)$$

where $\psi = \mathcal{H}_\Theta(e^i)$ and $\mathcal{B}^i$ is the query buffer for current task i. Minimizing the task-specific loss $\mathcal{L}^i_{\text{task}}$ updates the shared hypernetwork parameters Θ (and e^i) via backpropagation through the generated reward network. Additionally, in order to encapsulate the information regarding previous tasks, we regularize the hypernetwork output for all previous task embeddings. The regularizer is based on the following intuition: when adapting to the current teacher preferences, we update the hypernetwork such that its output on the previous task embeddings remains the same. Doing so ensures that the hypernetwork retains the ability to generate previous reward network parameters even after learning a new task. We checkpoint the hypernetwork before learning task i ($\mathcal{H}_{\Theta^{i-1}}$). Using $\mathcal{H}_{\Theta^{i-1}}$, the current hypernetwork, $\Delta\Theta$, and the embeddings, we define a regularization loss for each previous task $1 \leq j \leq i-1$ as

$$\mathcal{L}^{i,j}_{\text{reg}}\big(e^j, \Theta^{i-1}, \Delta\Theta; \Theta\big) = \big\|\mathcal{H}(e^j, \Theta^{i-1}) - \mathcal{H}(e^j, \Theta + \Delta\Theta)\big\|^2_2, \quad (3)$$

where $\|\cdot\|_2$ denotes the L^2 norm, and Θ are the current (task i) hypernetwork parameters.

Lookahead: The lookahead, $\Delta\Theta$ is a candidate change in the hypernetwork parameters that minimizes the loss for the current task $\mathcal{L}^i_{\text{task}}$ with respect to Θ (Eq. 2). Its purpose is to provide a lookahead (preview) about the gradient direction (if minimizing only the current task loss function) before taking a gradient step. To avoid higher-order derivatives and computational overhead, we approximate $\Delta\Theta$ using a one-step lookahead via a single Adam update (Kingma & Ba, 2017) and detach it from the computational graph. We observe that this performs well empirically (Section 5).

The term in Equation 3 aids in remembering how to solve previous tasks, consistent with idea of the regularization-based methods as categorized by Van de Ven et al. (2022). However, the above term is different from conventional regularization in that it operates in the output space of the hypernetwork as opposed to the its parameter space. We do not constrain the hypernetwork's parameters Θ to remain close to their previous values, resulting in greater flexibility, as we shall see later. Aggregating the regularization losses over all previous tasks, we get $\mathcal{L}^i_{\text{reg}} = \frac{1}{i-1}\sum_{j=1}^{i-1}\mathcal{L}^{i,j}_{\text{reg}}$, for all $i > 1$.

Finally, the total (dual) loss function combines the current task loss and the regularization loss as:

$$\mathcal{L}^i_{\text{total}} = \begin{cases} \mathcal{L}^i_{\text{task}}(\sigma^0, \sigma^1, y; \Theta, e^i) + \beta_{\text{reg}} \mathcal{L}^i_{\text{reg}}\big(e^{<i}, \Theta^{i-1}, \Delta\Theta; \Theta\big) & i > 1 \\ \mathcal{L}^i_{\text{task}}(\sigma^0, \sigma^1, y; \Theta, e^i) & i = 1 \end{cases}, \tag{4}$$

where β_{reg} controls the strength of the regularizer, $i > 0$ denotes the current task, and all other terms are defined as before. Note that although $\mathcal{L}^i_{\text{total}}$ is a function of all embeddings thus far, only the current embedding e^i is updated; the previous embeddings $e^{<i}$ are not changed. This gives the hypernetwork the flexibility to find (near–) optimal reward functions for both previous tasks and current task i. Optimizing $\mathcal{L}^i_{\text{task}}$ aids in plasticity by allowing Θ and e^i to adapt to perform well at task i. Simultaneously, optimizing $\mathcal{L}^i_{\text{reg}}$ improves stability by preserving past knowledge while updating Θ, hence avoiding catastrophic forgetting. β_{reg} is a tunable hyperparameter acting as a bridge between both losses. Note that computing the total loss requires only the current task's preference data. The regularizer requires the current Θ and the checkpointed Θ^{i-1} (reset after each task), which is fixed memory. Consistent with prior work by Von Oswald et al. (2019), our regularizer requires the frozen past $e^{<i}$. Hence, we save the trained embedding after each task. Although this increases the memory requirements with tasks, each embedding is a self-learned low-dimensional vector (Table 8), which is several times cheaper than teacher-dependent past preference data typically entailing thousands of multi-step trajectory segments. For many applications, this is a minimal overhead.

4.2 Overall Algorithm

Algorithm 1 describes the continual training procedure entailing sequential tasks with task-specific teacher preferences. The algorithm follows a bi-level structure with the outer loop over tasks. We initialize an empty buffer $\mathcal{E}$ to save the learned embeddings; a fresh task-specific embedding e^i is initialized for each new task. For optimizing a policy over the task-specific reward, we reset the actor-critic networks and empty replay and query buffers (Line 4), which is a fixed amount of memory independent of the number of tasks. In the inner loop, the following two processes occur:

- Policy Updates: The agent interacts with the environment, stores experience in the replay buffer, $\mathcal{D}^i$ (labeled with the current reward estimate r^i_ψ), and updates the policy via Soft Actor-Critic (Haarnoja et al., 2018) (Lines 15–16).
- Reward Updates: Every K steps, the agent queries the teacher for preferences on trajectory segments sampled from $\mathcal{D}^i$. The segments and corresponding labels are stored in B^i. The hypernetwork's Θ and e^i are then updated to minimize the dual loss $\mathcal{L}^i_{\text{total}}$ (Eq. 4 and Lines 7–13).

An important step in the algorithm is relabeling the entire replay buffer with the updated reward function (Lee et al., 2021a). This ensures the policy is always optimizing against the most recent reward model (Line 13). Finally, we conclude the task loop by saving the learned embedding e^i for future use and the current state of the hypernetwork is checkpointed. A visual representation of Algorithm 1 is provided in Figure 16 (Appendix 15).

5 Experiments and Results

We perform the experiments to investigate the following research questions:

- **RQ1:** Can HNET (Algorithm 1) balance the dual objectives of learning new tasks and retaining performance on previously learned tasks? How well does it transfer knowledge forward? How does it compare to other continual learning baselines adapted to the CPRL setting?

  We evaluate the quality of the learned reward function via the performance of policy optimized over it. Specifically, we optimize a policy over the learned reward model (using SAC) and measure the policy performance in two phases, continual training and knowledge retention, each covering individual aspects of the dual objectives. The details of these metrics are provided in Section 5.1.

Algorithm 1 Continual Preference-based RL with Hypernetworks (CPRL with HNET)

Input: regularization strength β_{reg}, budget B, frequency K
Initialize: hypernetwork $\mathcal{H}_\Theta$, embedding buffer $\mathcal{E} \leftarrow \emptyset$, checkpoint $\mathcal{H}_{\Theta^0} \leftarrow \mathcal{H}_\Theta$
for task $i = 1, 2, \ldots$ **do**
  Initialize task embedding e^i, reset buffers $\mathcal{D}^i, \mathcal{B}^i$, and actor-critic networks (π^i_ϕ, Q^i_ω)
  Generate initial reward parameters $\psi = \mathcal{H}_\Theta(e^i)$
  for each iteration **do**
    if iteration $\% \, K == 0$ **and** collected queries $< B$ **then**
      Sample trajectory segments from $\mathcal{D}^i$, query the teacher for labels, and add to $\mathcal{B}^i$
      repeat
        Compute loss $\mathcal{L}^i_{\text{total}} = \mathcal{L}^i_{\text{task}}(\mathcal{B}^i) + \beta_{\text{reg}}\mathcal{L}^i_{\text{reg}}(\mathcal{E}, \mathcal{H}_{\Theta^{i-1}})$
        Update Θ and e^i via `Adam` on $\mathcal{L}^i_{\text{total}}$
      until reward accuracy converges
      Update reward parameters $\psi \leftarrow \mathcal{H}_\Theta(e^i)$ and relabel entire $\mathcal{D}^i$ using r^i_ψ
    end if
    Collect experience (s, a, s') via π^i_ϕ; compute $r = r^i_\psi(s_t, a_t)$; store (s, a, r, s') tuples in $\mathcal{D}^i$
    Update π^i_ϕ and Q^i_ω using SAC
  end for
  Store e^i in $\mathcal{E}$; checkpoint the hypernetwork $\mathcal{H}_{\Theta^i} \leftarrow \mathcal{H}_\Theta$
end for

- **RQ2:** Does HNET learn reward functions that remain consistently aligned with the teacher's underlying preferences across tasks, beyond what is reflected by policy performance alone?

  Reward Alignment: Relying solely on policy performance to measure quality of the learned reward can be misleading: a policy might fail due to an instability in its optimizer, even if the reward model is perfect. Conversely, a policy might achieve high returns by exploiting a specific behavior without the learned reward being aligned with the teacher's preferences (Gleave et al., 2021). Therefore, we use an additional metric given by the *trajectory alignment coefficient*, $\tau_{\text{tac}} \in [-1, 1]$ (Muslimani et al., 2025). τ_{tac} measures the level of agreement between two sets of trajectory preference orderings. Here, we use it to measure alignment between the teacher's preferences over trajectories and those induced by learned reward function. The details of τ_{tac} are in Appendix 16.3.

5.1 Experimental Setup

Environments We first adapt the following classic control, manipulation, and locomotion environments by varying goals across the task sequence.

- **Gymnasium Pendulum-v1:** A sequence of 9 tasks where the objective is to align the pendulum to changing target angles with the vertical (by applying torque against gravity) and stay there.
- **Meta-World Reach-v2:** A 4-task sequence, where the objective is to reach (i.e., move the robotic end-effector) different target locations in the 3-D space (Yu et al., 2021).
- **MuJoCo HalfCheetah:** A 3-task sequence, where the objective is to make the cheetah run with different target velocities. The implementation is taken from Gymnasium (Towers et al., 2024).

These environments evaluate the performance when the dynamics remain same but the reward function (preferences) changes. The exact numerical target values (that define the preferences and hence the reward) for these sequences are specified in Table 5 in Appendix 16. The number of tasks in a sequence is chosen such that the coverage is sufficient to demonstrate balancing the dual objectives.

We also consider sequences of *distinct* Meta-World tasks. We construct two 3-task sequences to test forward transfer and knowledge retention. Forward transfer refers to transferring knowledge to

future tasks due to experience on previous tasks. Here, tasks $1 \rightarrow 3$ have a higher transfer potential than tasks $2 \rightarrow 3$. Hence, the agent needs to not forget for positive transfer (Wołczyk et al., 2021).

- **Reaching Sequence:** [reach-v2 $\rightarrow$ drawer-open-v2 $\rightarrow$ reach-wall-v2]. Here, the first and third tasks share the mechanics of reaching towards a target, while the intermediate task introduces complex manipulation dynamics but still requires transferring the skill related to reaching a target.
- **Pressing Sequence:** [button-press-v2 $\rightarrow$ faucet-close-v2 $\rightarrow$ drawer-close-v2]. These tasks are contact-heavy and more challenging than the reaching sequence. The first and third tasks focus on a pushing/pressing motion, separated by a rotational manipulation task.

These sequences challenge the agent to remember and transfer skills from the first task to the third, despite the interference potential from the distinct second task. We model the HalfCheetah sequence using a similar design (Table 5).

Simulated Teacher We use a simulated teacher based on the ground-truth (noise-free) reward r^{gt}, as commonly done in PbRL (Lee et al., 2021a). We limit the feedback budget for each task: 200 queries per task for Pendulum, 2000 queries per task for Reach-v2 and HalfCheetah environments, and 2500 queries per task for the Reaching and Pressing sequences. The budget is chosen such that any inability to learn new tasks or remember previous tasks is not due to an insufficient budget. The queries are generated uniformly from the collected trajectory segments. The simulated teacher allows using r^{gt} to evaluate the agent quantitatively (policy returns and τ_{tac}) because if r^{gt} is used to generate the preference labels, learning from them ought to achieve high returns according to r^{gt}.

Baselines:

1. Oracle: The oracle trains a policy (using SAC) directly on r^{gt}. This enables assessing an unachievable yet interesting performance benchmark, which serves as a helpful standard.
2. Learning from Scratch: "Scratch" learns each task in the sequence from scratch by reinitializing the reward model (implemented using the hypernetwork architecture for fair comparison) for every new task. The agent learns each task independently and does not attempt to retain previous knowledge. This allows assessing whether a sequence is interesting (and non-trivial) enough such that learning just the recent-most task is insufficient to ensure good performance on previous tasks.
3. Synaptic Intelligence (SI) (Zenke et al., 2017): SI is a regularization-based method to balance learning new and remembering old tasks. It maintains an importance measure for each parameter based on its contribution to the drop in loss during training. When learning a new task, a quadratic penalty is added to the loss to discourage changing parameters deemed important for past tasks.
4. Deep-Generative Replay (DGR) (Shin et al., 2017): DGR is a replay-based method using data generation to balance the dual objectives. To avoid storing real trajectory data, we train a Variational AutoEncoder (VAE) to generate synthetic data (trajectory segments) representative of past tasks. During reward learning, current reward model is trained on a mixture of real feedback from the current teacher and generated segments labeled by a snapshot of the previous reward model.

We evaluate the baselines with multiple configurations. For SI, we vary the regularization strength ($\beta \in \{0.1, 1.0, 10.0\}$), which controls the balance between learning new and remembering previous tasks. For DGR, we adjust the VAE regularization coefficient ($\lambda \in \{1.0, 5.0\}$), which dictates the quality and regularity of the synthesized trajectories used for replay. Including these variants, as we shall see, allows demonstrating that the better performance of HNET is not merely due to suboptimal tuning of the baselines, but represents a fundamental improvement. Orthogonal to the above baselines, we have the representation learning-based CL methods such as Continual Backpropagation (Dohare et al., 2024). We do not consider these for current evaluation because they are more focused towards loss of plasticity and do not explicitly consider knowledge retention.

Policy Performance Metrics In order to evaluate the policy performance at each measurement step, we pause the learning to run 15 episodes using the current policy and report the mean of ground-truth undiscounted returns (Patterson et al., 2024).

Domain	Metric	Oracle	Scratch	SI	DGR	HNET
Pendulum-v1	Continual training	-55.5 ± 7.1	-193.4 ± 44.7	-1811.8 ± 215.4	-211.0 ± 51.1	$\mathbf{-210.9\pm58.6}$
	Catastrophic forgetting	-55.5 ± 7.1	-3236.1 ± 126.1	-2739.9 ± 155.4	-766.1 ± 188.1	$\mathbf{-261.6\pm67.2}$
Reach-v2	Continual training	4863.6 ± 16.8	4860.5 ± 20.7	3664.2 ± 374.0	4862.9 ± 18.8	$\mathbf{4871.7\pm21.6}$
	Catastrophic forgetting	4864.3 ± 16.4	1770.4 ± 168.2	1546.5 ± 209.0	2591.3 ± 241.5	$\mathbf{4540.9\pm252.5}$
Half Cheetah	Continual training	-72.0 ± 2.7	$\mathbf{-670.1\pm58.9}$	-912.3 ± 175.2	-974.5 ± 133.5	-722.4 ± 122.1
	Catastrophic forgetting	-72.0 ± 2.9	-1814.0 ± 86.8	-1985.4 ± 156.4	-1417.2 ± 199.4	$\mathbf{-1075.7\pm115.7}$
Reaching Sequence	Continual training	4170.2 ± 127.3	3947.0 ± 134.4	3779.6 ± 195.7	3989.3 ± 149.2	$\mathbf{4064.2\pm36.6}$
	Catastrophic forgetting	4197.5 ± 162.4	1656.4 ± 291.6	1897.0 ± 385.3	2231.0 ± 382.3	$\mathbf{3423.4\pm453.1}$
Pressing Sequence	Continual training	4055.6 ± 77.7	3192.4 ± 229.6	3053.0 ± 542.9	3057.6 ± 445.6	$\mathbf{3443.1\pm456.3}$
	Catastrophic forgetting	4050.1 ± 76.3	571.6 ± 315.2	$\mathbf{2453.9\pm386.3}$	610.1 ± 274.0	2334.9 ± 518.9

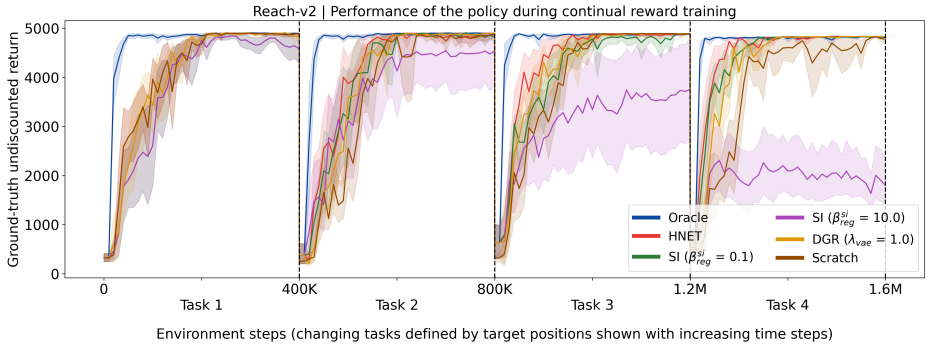
Table 1: Comparison of continual training and forgetting performance in terms of the consolidated metric across domains. Each entry reports mean $\pm$ confidence interval (95%) over 10 runs. Continual training performance measures the final return (averaged over all tasks) during reward training, while catastrophic forgetting measures the average retained performance when using remembered reward function on previous tasks. *Higher is better*. For HNET, SI, and DGR, we report the results using hyperparameters that best balance the two metrics of learning new knowledge and preserving past knowledge. The objective is to achieve this balance (**RQ1**). Note: Oracle and Scratch are calibration baselines. The best mean performance (excluding Oracle) is in **bold**.

1. *Continual training:* This measures how well the agent learns each new task (i.e., reward function reflecting preferences) during continual training in Algorithm 1. We also report a single consolidated metric for training performance in Table 1. For each random seed (run), we take the final recorded return for each task and average across tasks to form a per-run score (AUC scores are in Appendix 10). We report the group statistic as the mean across runs with confidence intervals.
2. *Catastrophic forgetting:* This involves evaluating how well the hypernetwork can generate previous task reward functions after continual training (end of task sequence of length N). For a past task k, we retrieve its (frozen) embedding $e^k \sim \mathcal{E}^N$ and pass it through the trained hypernetwork $\mathcal{H}^N_\Theta$ to obtain the (remembered) reward parameters for task k. Next, we initialize a new SAC agent and learn a policy from the remembered reward function (no reward updates occur here). The learned policy should perform well if the (parameterized) reward function was remembered well, owing to the regularizer. Hence, policy performance gives us an estimate of forgetting. This procedure is explained in Algorithm 2 in the appendix with a visual representation in Figure 17. For other baselines, we similarly obtain the (remembered) reward model and test forgetting. We also report a consolidated metric as done before except that there is no reward training here.
3. *Forward Transfer:* This involves measuring the effect (positive/negative) of knowledge acquired from previous tasks when learning a new task. We compute forward transfer as the AUC of the concerned method relative to learning task from scratch. For details, please refer Appendix 14.

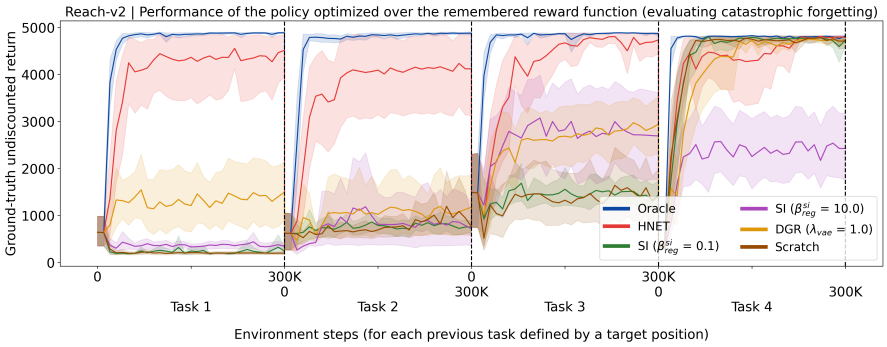
5.2 Results

RQ1: We report the consolidated metrics for the dual objectives in Table 1 and forward transfer in Table 2. In Figures 2 and 3, we present the learning curves for Reach-v2. The results for other domains are in Appendix 9. All the results are averaged over 10 runs with 95% confidence intervals.

Continual training & catastrophic forgetting: Recall that we want to balance the dual objectives evaluated by these two metrics (Section 1). Hence, we analyze both together. To summarize the results in Table 1, HNET generally performs better than (or similar to) SI, DGR, and Scratch in terms of policy performance for balancing the dual objectives. Further, the HNET performance is worse than Oracle and the difference provides an informative benchmark. There can be cases when a baseline performs similar to HNET for a particular metric: consider the Scratch performance regarding continual training in Table 1. However, HNET outperforms Scratch in retaining past knowledge. This is because Scratch, by construction (Section 5.1) learns each new task well but fails to remember any previous task. Similarly, SI with a weak regularization can perform similar to HNET during continual training, but this comes at the cost of forgetting.



(a) Performance of the policy during continual learning of reward functions (tasks) in Reach-v2. The curves indicate whether close-to-oracle performance can be achieved on each sequential task.



(b) Performance of the policy (on each previous task) obtained by optimizing over the (learned) rewards $r^N_{\psi_k}$ $(1 \leq k \leq N)$ in Reach-v2. The reward has been trained continually for $N = 4$ tasks. The policy performance curves (indirectly) indicate how well the memory of previous tasks' reward functions is retained. The x-axis is discontinuous, representing policy optimization on N independent tasks without reward learning.

Figure 2: Performance of HNET and baselines on Meta-World Reach-v2 during continual reward training (a) and forgetting evaluation (b). This result shows that HNET (ours) outperforms SI, DGR, and Scratch in terms of learning new and preserving old knowledge in Reach-v2.

When considering the dual objectives together, HNET outperforms the baselines (except Oracle) in Pendulum-v1, Reach-v2, and the Reaching Sequence. In HalfCheetah, HNET outperforms the baselines in forgetting and performs competitively in continual training. In the Pressing Sequence, HNET outperforms DGR and Scratch in forgetting and performs competitively with SI. For continual training, the performance across algorithms is similar.

Figure 2 shows the learning curves for Reach-v2 that demonstrate the inability of SI, DGR, and Scratch to balance learning new and remembering old reward functions in Reach-v2 and the superior performance of HNET. The learning curves for other domains are shown in Figures 5, 7, 9, 11. Note that for the forgetting evaluation plots (Figure 2b), the x-axis is discontinuous, representing learning N independent tasks and no continual reward learning. The x-axis ticks are represented in a two-line format. The starting x-tick of each task (except the first) is the same as the ending x-tick of previous task. Hence, the x-tick time-steps for tasks $1, 3, 5, \ldots$ are arranged at the top and for tasks $2, 4, 6, \ldots$ at the bottom. For statistical significance tests, please check Appendix 11.

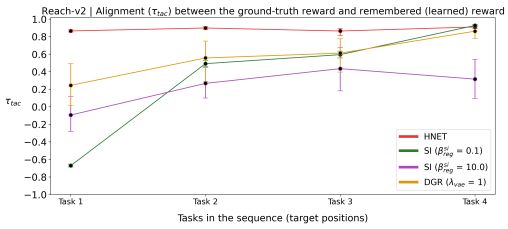
Analysis: We analyze a few important facets of the results. In HalfCheetah, the performance (in general) is poor compared to the Oracle across all algorithms. We hypothesize that the reasons for this are the complexity in learning rewards and relatively uninformative trajectory segments in this locomotion environment (details in Appendix 9.2). The Pressing Sequence involves contact-rich dynamics and precise object manipulation unlike the kinematic reaching tasks. These lead to high sensitivity to random seeds across all baselines. We aim to improve these results in the future work.

Domain	Ours	Synaptic Intelligence (SI)			Deep Gen. Replay (DGR)	
	HNET	$\beta = 0.1$	$\beta = 1.0$	$\beta = 10.0$	$\lambda = 1.0$	$\lambda = 5.0$
HalfCheetah	$\mathbf{-0.03 \pm 0.40}$	-0.24 ± 0.65	-2.05 ± 0.93	-1.83 ± 0.95	-0.15 ± 0.12	-0.16 ± 0.23
Reaching Sequence	$\mathbf{+0.34 \pm 0.25}$	$+0.10 \pm 0.70$	-0.43 ± 0.88	-2.78 ± 0.20	$+0.13 \pm 0.46$	$+0.27 \pm 0.22$
Pressing Sequence	-0.07 ± 0.41	$\mathbf{+0.17 \pm 0.17}$	-0.49 ± 1.13	-0.73 ± 1.39	-0.01 ± 0.26	$+0.09 \pm 0.30$

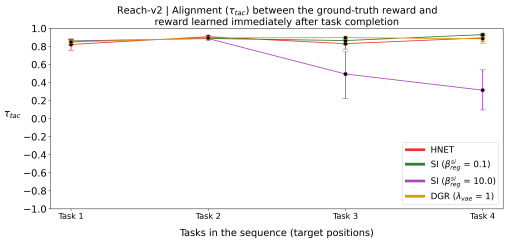
Table 2: Forward Transfer on the final task for each sequence. On the Reaching and Pressing sequences, FT measures the ability to transfer any knowledge from task 1 after interference from task 2 (Section 5.1). Values are unitless (formula to compute FT is in Appendix 14). *Higher is better*. Negative values indicate interference. HNET generally shows positive or neutral (≈ 0) forward transfer. The best mean performance is in **bold**.

Next, we hypothesize that HNET performing better than SI in many cases is because SI constrains the model directly by penalizing deviations from the parameters deemed relevant for previous tasks. This requires trading-off performance on some tasks to improve on other tasks. Such a trade-off is prominent when the tasks are considerably different from each other. HNET sidesteps this trade-off by outputting task-specific models from a shared meta-model ($\mathcal{H}_\Theta$) due to the expressiveness of hypernetworks and flexibility of the regularizer. We establish utility of the regularizer via an ablation in Appendix 12. In case of DGR, the performance depends on quality of the VAE, which itself is trained continually and protected from forgetting by retraining on its own synthetic data. Even if the inaccuracies in the synthetic data are small, the errors might compound over the course of learning.

Forward transfer (FT): HNET shows either positive or neutral (≈ 0) FT (Table 2). This is in contrast to SI, which sometimes exhibits large negative transfer, indicating rigidity of parameters for new tasks (task 2 $\rightarrow$ task 3). DGR shows decent transfer capabilities, but is poorer than HNET in general. Table 2 indicates promising (yet not very strong) results for FT in HNET. We hypothesize that HNET allows segregating the learning of shared information and unique information in the task sequence in the form of shared hypernetwork parameters and task-specific embeddings, respectively. This promotes transferring the segregated shared information (task-to-task similarities) across tasks.



(a) Alignment for the remembered reward $r^N_{\psi_k}$ $(1 \leq k \leq N)$ that has been trained continually for $N = 4$ tasks in Reach-v2.



(b) Alignment for the reward $r^k_{\psi_k}$ $(1 \leq k \leq N)$ obtained immediately after finishing task k when training continually on Reach-v2 ($N = 4$).

Figure 3: Alignment between the ground-truth and learned rewards for tasks $1 \leq k \leq N$ for HNET and the baselines in Meta-World Reach-v2 ($N = 4$). This result shows that HNET learns rewards that are aligned better with the teacher's preferences than the baselines.

RQ2: We report the alignment (τ_{tac}) of the continually learned reward model with the r^{gt} (simulated teacher) in Figures 3a, 6a, 8a, 10a, 12a. The reward functions learned from HNET are consistently aligned with r^{gt} for each task, while the alignment is worse (or similar in HalfCheetah and the Pressing Sequence) for SI, DGR, and Scratch. This confirms that the better policy performance for HNET observed previously was indeed due to an aligned reward. Figures 3b, 6b, 8b, 10b, 12b report the alignment of the the reward function *during training* (i.e., immediately after finishing the respective task). Here, we observe consistent alignment with r^{gt} for all algorithms. This is expected because the reward has been learned recently. The above results also provide insights into the level of alignment (and quality) of the reward functions learned by different methods. Such insights become relatively inconspicuous when only policy performance is observed.

6 Conclusion and Future Work

In this work, we introduced the framework of Continual Preference-based Reinforcement Learning to address the challenge of learning reward models from a sequence of non-stationary teacher preferences. We constructed a hypernetwork-based algorithm that utilizes task-specific embeddings and output-space regularization to balance the retention of past knowledge with the adaptation to new tasks. Our experiments across five domains demonstrate that this approach performs better (or competitively with) traditional regularization and replay-based baselines, maintaining high reward alignment and facilitating stable policy learning. For future work, we plan to experiment with noisy, delayed, and irregular feedback from humans over the noise-free r^{gt}-based simulated teacher. We also plan to experiment with longer task sequences and integrating HNET with gradient descent variants suited to balancing plasticity and forgetting (Elsayed & Mahmood, 2023).

7 Author Contributions

V.R.N. conceived the core ideas, designed the algorithm, wrote the codebase, and executed empirical evaluations in two RL environments. S.S. extended the experiments to include three more environments. C.M. provided mentorship and was involved in the discussion and design of experiments, particularly reward alignment. S.D. and M.E.T. provided overall supervision, resources, and project administration. All authors contributed to writing the manuscript and approved the final version. *Note*: a major portion of this work was part of V.R.N.'s MS thesis at the University of Alberta.

Acknowledgments

Part of this work has taken place in the Intelligent Robot Learning (IRL) Lab at the University of Alberta, which is supported in part by research grants from the Alberta Machine Intelligence Institute (Amii), a Canada CIFAR AI Chair, the Digital Research Alliance of Canada, and the Natural Sciences and Engineering Research Council (NSERC). We thank all supporting bodies for their funding and resources.

References

Dario Amodei, Chris Olah, Jacob Steinhardt, Paul Christiano, John Schulman, and Dan Mané. Concrete problems in ai safety. *arXiv preprint arXiv:1606.06565*, 2016.

Sayantan Auddy, Jakob Hollenstein, Matteo Saveriano, Antonio Rodríguez-Sánchez, and Justus Piater. Continual learning from demonstration of robotics skills. *Robotics and Autonomous Systems*, 165:104427, 2023.

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

Gail A Carpenter and Stephen Grossberg. A massively parallel architecture for a self-organizing neural pattern recognition machine. *Computer vision, graphics, and image processing*, 37(1): 54–115, 1987.

Vinod Kumar Chauhan, Jiandong Zhou, Ping Lu, Soheila Molaei, and David A Clifton. A brief review of hypernetworks in deep learning. *Artificial Intelligence Review*, 57(9):250, 2024.

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Shibhansh Dohare, J Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, A Rupam Mahmood, and Richard S Sutton. Loss of plasticity in deep continual learning. *Nature*, 632(8026): 768–774, 2024.

Mohamed Elsayed and A Rupam Mahmood. Utility-based perturbed gradient descent: An optimizer for continual learning. *arXiv preprint arXiv:2302.03281*, 2023.

Adam Gleave, Michael Dennis, Shane Legg, Stuart Russell, and Jan Leike. Quantifying differences in reward functions. In *International Conference on Learning Representations*, 2021.

David Ha, Andrew M. Dai, and Quoc V. Le. Hypernetworks. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=rkpACe1lx.

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

Yizhou Huang, Kevin Xie, Homanga Bharadhwaj, and Florian Shkurti. Continual model-based reinforcement learning with hypernetworks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 799–805. IEEE, 2021.

Zhengran Ji and Boyuan Chen. Pref-guide: Continual policy learning from real-time human feedback via preference-based learning. *arXiv preprint arXiv:2508.07126*, 2025.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

Kimin Lee, Laura Smith, and Pieter Abbeel. Pebble: Feedback-efficient interactive reinforcement learning via relabeling experience and unsupervised pre-training, 2021a. URL https://arxiv.org/abs/2106.05091.

Kimin Lee, Laura Smith, Anca Dragan, and Pieter Abbeel. B-pref: Benchmarking preference-based reinforcement learning. *arXiv preprint arXiv:2111.03026*, 2021b.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Calarina Muslimani, Kerrick Johnstonbaugh, Suyog Chandramouli, Serena Booth, W. Bradley Knox, and Matthew E. Taylor. Towards improving reward design in rl: A reward alignment metric for rl practitioners, 2025. URL https://arxiv.org/abs/2503.05996.

Jongjin Park, Younggyo Seo, Jinwoo Shin, Honglak Lee, Pieter Abbeel, and Kimin Lee. Surf: Semi-supervised reward learning with data augmentation for feedback-efficient preference-based reinforcement learning. *arXiv preprint arXiv:2203.10050*, 2022.

Andrew Patterson, Samuel Neumann, Martha White, and Adam White. Empirical design in reinforcement learning. *Journal of Machine Learning Research*, 25(318):1–63, 2024. URL http://jmlr.org/papers/v25/23-0183.html.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. *Advances in neural information processing systems*, 30, 2017.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

Gido M Van de Ven, Tinne Tuytelaars, and Andreas S Tolias. Three types of incremental learning. *Nature Machine Intelligence*, 4(12):1185–1197, 2022.

Johannes Von Oswald, Christian Henning, Benjamin F Grewe, and João Sacramento. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019.

Christian Wirth, Riad Akrour, Gerhard Neumann, and Johannes Fürnkranz. A survey of preference-based reinforcement learning methods. *Journal of Machine Learning Research*, 18(136):1–46, 2017.

Maciej Wołczyk, Michał Zając, Razvan Pascanu, Łukasz Kuciński, and Piotr Miłoś. Continual world: A robotic benchmark for continual reinforcement learning, 2021. URL https://arxiv.org/abs/2105.10919.

Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2021. URL https://arxiv.org/abs/1910.10897.

Friedemann Zenke, Ben Poole, and Surya Ganguli. Continual learning through synaptic intelligence. In *International conference on machine learning*, pp. 3987–3995. PMLR, 2017.

Han Zhang, Yu Lei, Lin Gui, Min Yang, Yulan He, Hui Wang, and Ruifeng Xu. Cppo: Continual learning for reinforcement learning with human feedback. In *The Twelfth International Conference on Learning Representations*, 2024.

Han Zhang, Lin Gui, Yu Lei, Yuanzhao Zhai, Yehong Zhang, Zhuo Zhang, Yulan He, Hui Wang, Yue Yu, Kam-Fai Wong, et al. Copr: Continual human preference learning via optimal policy regularization. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 5377–5398, 2025.

Supplementary Materials

The following content was not necessarily subject to peer review.

8 Catastophic Forgetting with Fine-Tuning

Recall that we claimed in Section 1 that fine-tuning, i.e., training the model incrementally on each task using only the loss on the current data generally suffers from catastrophic forgetting.

To demonstrate this phenomenon, we train an agent with PEBBLE (Lee et al., 2021a), a popular PbRL algorithm, on a sequence of three robotic manipulation tasks without reinitializing the reward network parameters between tasks. After continual training on all three tasks, we evaluate performance on each task individually. We find that PEBBLE performs well only on the final task, indicating forgetting in the first two tasks. This occurs because updating the model to fit new preferences overrides parameters learned from earlier ones, leading to catastrophic forgetting (Fig. 4).[1]

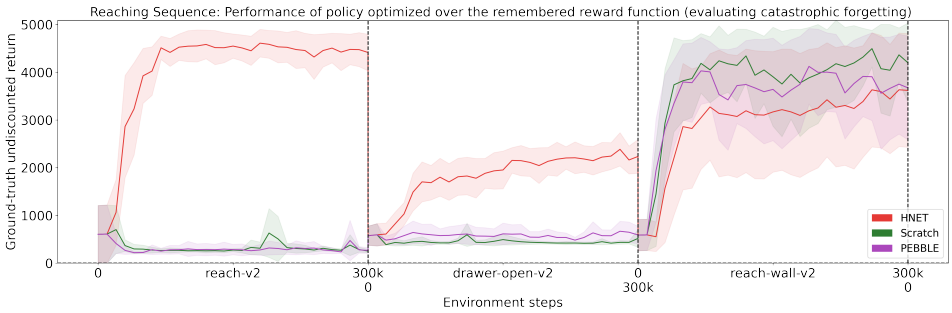


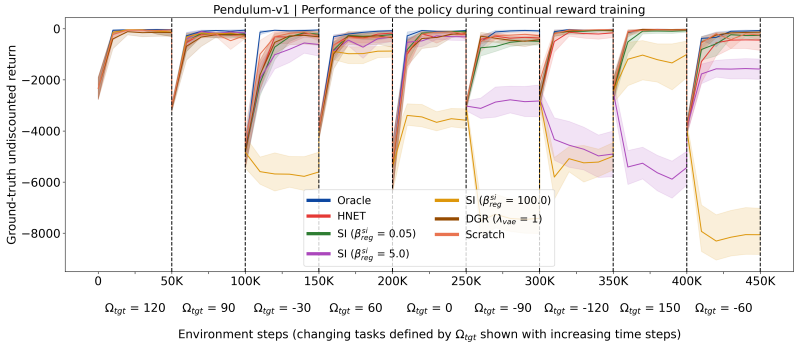
Figure 4: Performance of policy (on each previous task) obtained by optimizing over the (learned) reward model. The reward has been trained continually on three Meta-world tasks (Yu et al., 2021). The reward model learned using PEBBLE (without reward network reinitialization between tasks) retains the preferences only for the most recent task 3 (reach-wall-v2) and catastrophically forgets the first two tasks. In contrast, our method (HNET), outperforms PEBBLE and Scratch (re-initializing hypernetwork at the beginning of each task).

9 Learning curves for Pendulum, HalfCheetah, Reaching sequence, and Pressing sequence environments

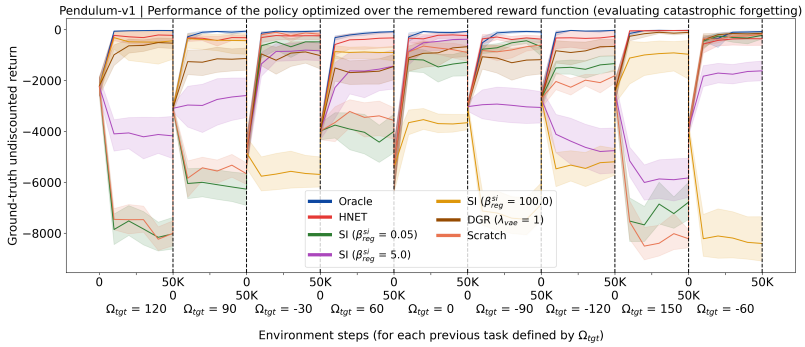
9.1 Pendulum-v1

Following are the learning curves for the Pendulum-v1 domain.

[1] More details are provided in the ablation in Appendix 12.

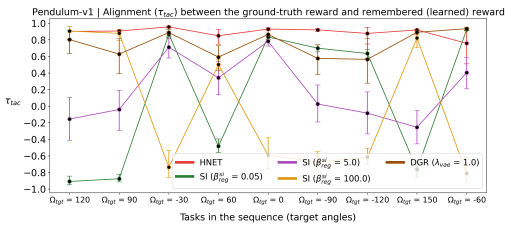


(a) Performance of the policy during continual learning of reward functions (tasks) in Pendulum. The curves indicate whether close-to-oracle performance can be achieved on each sequential task.

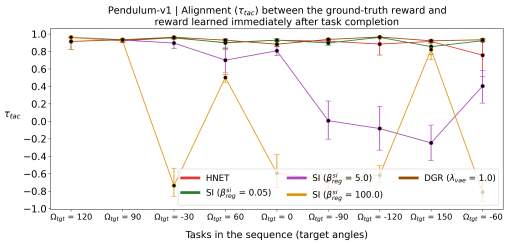


(b) Performance of the policy (on each previous task) obtained by optimizing over the (learned) rewards $r^N_{\psi_k}$ $(1 \leq k \leq N)$ in Pendulum. The reward has been trained continually for $N = 9$ tasks. The policy performance curves (indirectly) indicate how well the memory of previous tasks' reward functions is retained. The x-axis is discontinuous, representing policy optimization on N independent tasks without reward learning.

Figure 5: Performance of HNET and baselines on Pendulum during continual reward training (a) and forgetting evaluation (b). This result shows that HNET (ours) outperforms SI, DGR, and Scratch in terms of learning new and preserving old knowledge in the Pendulum domain.



(a) Alignment for the remembered reward $r^N_{\psi_k}$ $(1 \leq k \leq N)$ that has been trained continually for $N = 9$ tasks in Pendulum.

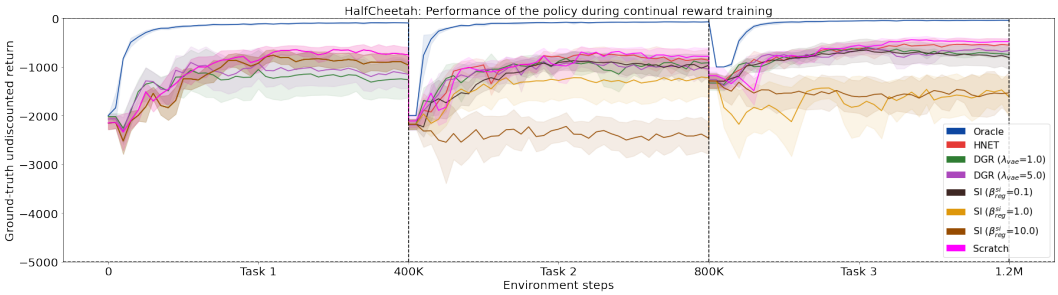


(b) Alignment for the reward $r^k_{\psi_k}$ $(1 \leq k \leq N)$ obtained immediately after finishing task k when training continually on Pendulum ($N = 9$).

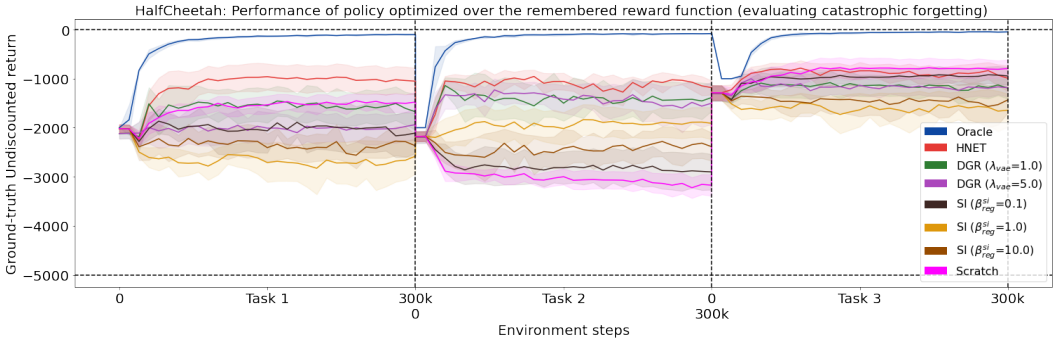
Figure 6: Alignment between the ground-truth and learned rewards for tasks $1 \leq k \leq N$ for HNET and the baselines in Pendulum ($N = 9$). This result shows that HNET learns rewards that are aligned better with the teacher's preferences than the baselines.

9.2 HalfCheetah

Following are the learning curves for the HalfCheetah domain.



(a) Performance of the policy during continual learning of reward functions (tasks) in HalfCheetah. The curves indicate whether close-to-oracle performance can be achieved on each sequential task.



(b) Performance of the policy (on each previous task) obtained by optimizing over the (learned) rewards $r^N_{\psi_k}$ $(1 \leq k \leq N)$ in HalfCheetah. The reward has been trained continually for $N = 3$ tasks. The policy performance curves (indirectly) indicate how well the memory of previous tasks' reward functions is retained. The x-axis is discontinuous, representing policy optimization on N independent tasks without reward learning.

Figure 7: Performance of HNET and baselines on HalfCheetah during continual reward training (a) and forgetting evaluation (b).

Recall that we mentioned in Section 5.2 that the performance of all algorithms is somewhat worse compared to the Oracle on HalfCheetah. We provide a detailed explanation here.

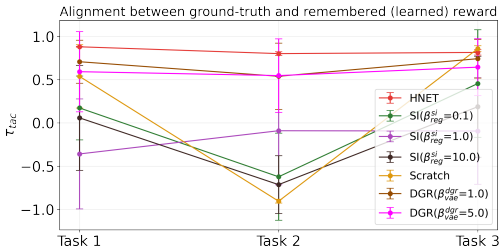
First, we would like to clarify that this is a modified HalfCheetah environment that performs velocity tracking (achieving a desired velocity given by the teacher's preferences) and not velocity maximization. Therefore, the theoretical maximum return can only be 0 (based on the ground-truth reward), which the oracle closes on to.

Regarding why other algorithms do not achieve Oracle performance, we believe that this is due to the difficulty of learning rewards inherent to the HalfCheetah environment. In Meta-World (Reaching/Pressing), the ground-truth reward is heavily dominated by Euclidean distance (e.g., distance from the end-effector to the goal). We found that the reward network efficiently learns this "closer is better" relationship from preferences in terms of the distance. On the other hand, HalfCheetah is a complex locomotion task (state dim $= 17$, action dim $= 6$). Making the velocity of the cheetah robot similar to the desired velocity does not necessarily translate the same as moving one object near another object spatially (like in Meta-World). The dynamics are very different and more complex here.

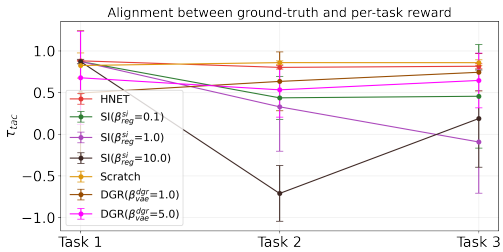
Another reason for this gap could potentially be attributed to the temporally short trajectory segments. In our preference learning implementation, a preference label is obtained over a pair of trajectory segments, which are a subset of the full trajectory. In case of reacher and pendulum environments, even having subsets of the full trajectory along with a preference label would indicate

the direction in which the teacher intends the agent to go. Hence, the preference queries are relatively noise-free. In case of HalfCheetah, which is a locomotion environment, a short segment might capture the robot mid-stride or momentarily while it loses balance to gain forward momentum. Consequently, the preference signal over a short temporal window is significantly noisier than it would be for other environments.

Due to the above two reasons, we think that reward learning in this environment is harder for all the methods. However, when comparing the relative performance, HNET significantly outperforms others in terms of catastrophic forgetting and performs similarly in continual training (Table 1).



(a) Alignment for the remembered reward $r^N_{\psi_k}(1 \leq k \leq N)$ that has been trained continually for $N = 3$ tasks in HalfCheetah.

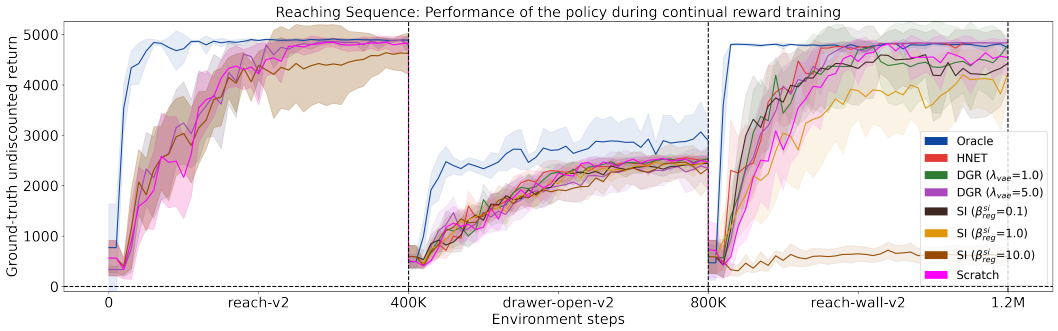


(b) Alignment for the reward $r^k_{\psi_k}(1 \leq k \leq N)$ obtained immediately after finishing task k when training continually on HalfCheetah ($N = 3$).

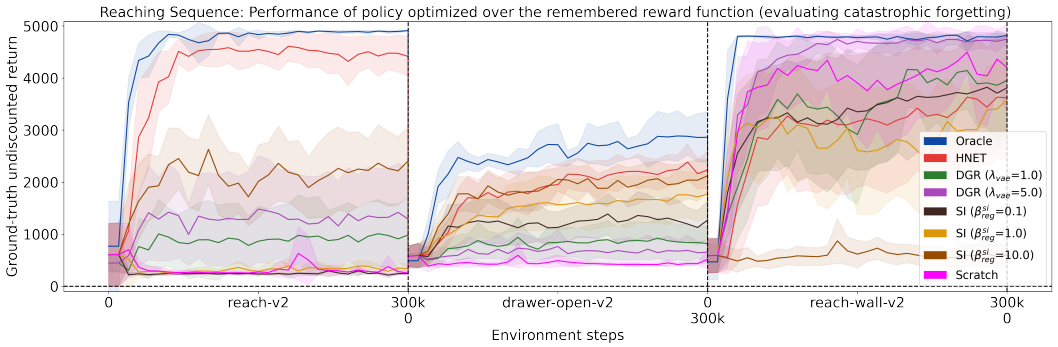
Figure 8: Alignment between the ground-truth and learned rewards for tasks $1 \leq k \leq N$ for HNET and the baselines in HalfCheetah ($N = 3$).

9.3 Meta-World Reaching Sequence

Following are the learning curves for the Meta-World Reaching Sequence.

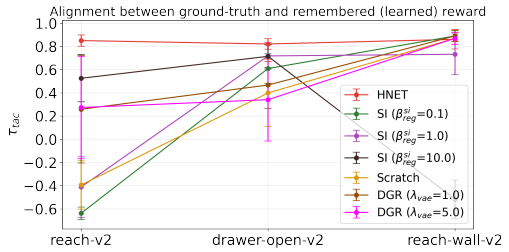


(a) Performance of the policy during continual learning of reward functions (tasks) in the Reaching Sequence. The curves indicate whether close-to-oracle performance can be achieved on each sequential task.

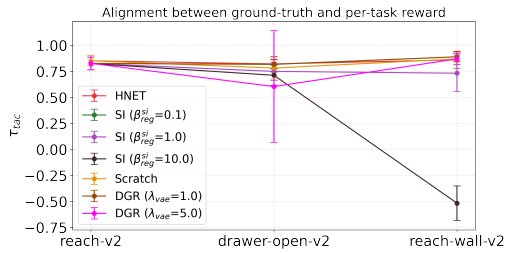


(b) Performance of the policy (on each previous task) obtained by optimizing over the (learned) rewards $r^N_{\psi_k}$ $(1 \leq k \leq N)$ in the Reaching Sequence. The reward has been trained continually for $N = 3$ tasks. The policy performance curves (indirectly) indicate how well the memory of previous tasks' reward functions is retained. The x-axis is discontinuous, representing policy optimization on N independent tasks without reward learning.

Figure 9: Performance of HNET and baselines on the Reaching Sequence during continual reward training (a) and forgetting evaluation (b). This result shows that HNET (ours) outperforms SI, DGR, and Scratch in terms of learning new and preserving old knowledge in Reaching Sequence.



(a) Alignment for the remembered reward $r^N_{\psi_k}$ $(1 \leq k \leq N)$ that has been trained continually for $N = 3$ tasks in the Reaching Sequence.

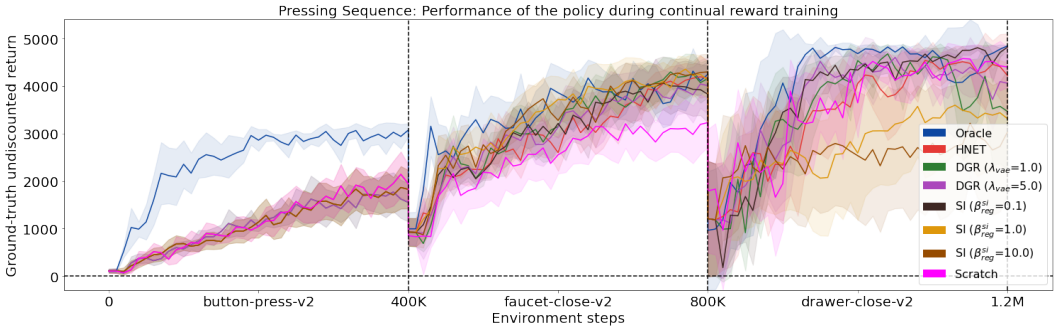


(b) Alignment for the reward $r^k_{\psi_k}$ $(1 \leq k \leq N)$ obtained immediately after finishing task k when training continually on the Reaching Sequence ($N = 3$).

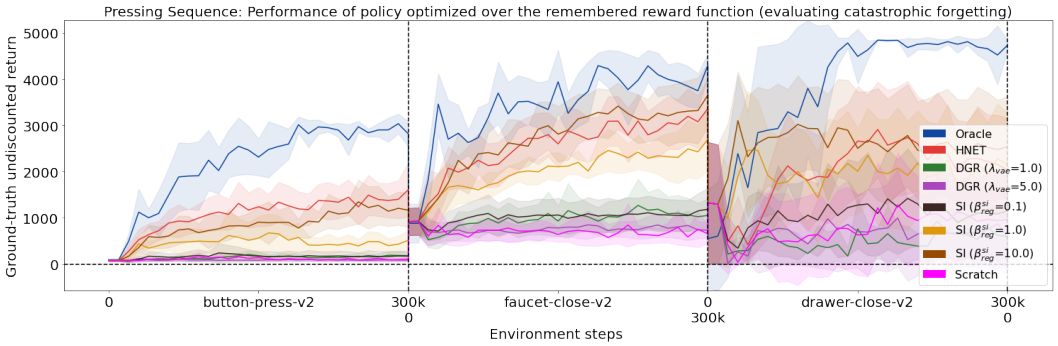
Figure 10: Alignment between the ground-truth and learned rewards for tasks $1 \leq k \leq N$ for HNET and the baselines in the Reaching Sequence ($N = 3$). This result shows that HNET learns rewards that are aligned better with the teacher's preferences than the baselines.

9.4 Meta-World Pressing Sequence

Following are the learning curves for the Meta-World Pressing Sequence.

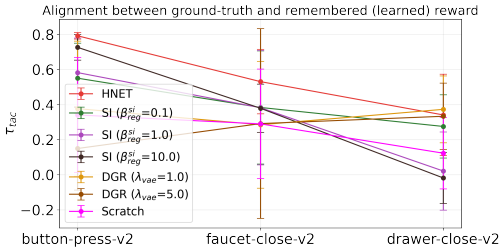


(a) Performance of the policy during continual learning of reward functions (tasks) in the Pressing sequence. The curves indicate whether close-to-oracle performance can be achieved on each sequential task.

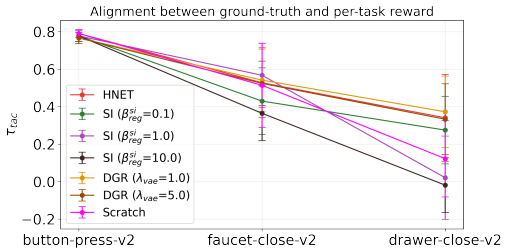


(b) Performance of the policy (on each previous task) obtained by optimizing over the (learned) rewards $r^N_{\psi_k}$ $(1 \leq k \leq N)$ in the Pressing sequence. The reward has been trained continually for $N = 3$ tasks. The policy performance curves (indirectly) indicate how well the memory of previous tasks' reward functions is retained. The x-axis is discontinuous, representing policy optimization on N independent tasks without reward learning.

Figure 11: Performance of HNET and baselines on the Pressing sequence during continual reward training (a) and forgetting evaluation (b).



(a) Alignment for the remembered reward $r^N_{\psi_k}$ $(1 \leq k \leq N)$ that has been trained continually for $N = 3$ tasks in the Pressing sequence.



(b) Alignment for the reward $r^k_{\psi_k}$ $(1 \leq k \leq N)$ obtained immediately after finishing task k when training continually on Pressing sequence ($N = 3$).

Figure 12: Alignment between the ground-truth and learned rewards for tasks $1 \leq k \leq N$ for HNET and the baselines in the Pressing sequence ($N = 3$).

9.5 Additional results based on the experiments

We discuss some noteworthy phenomena observed in our results (Table 1 and all the learning curves):

1. **Slight drop in performance on the final task:** We observe that HNET's performance on the most recently learned task (T_3) is marginally lower than that of some baselines (Figure 9b). This happens because baselines overfit to the current task, maximizing T_3 performance by catastrophically overwriting the weights tuned for T_1 and T_2. In contrast, HNET enforces an appropriate multi-objective constraint to perform well on all tasks. While this prevents it from dedicating its full capacity solely to T_3, this strategic trade-off is validated in Table 1: HNET achieves superior *average* knowledge retention, showing that avoiding overfitting on the recent-most task may be essential for overall knowledge retention.
2. **Severe lack of forward transfer for SI on HalfCheetah:** SI exhibits catastrophic interference ($FT \approx -2.0$) on the HalfCheetah sequence because the regularization mechanism actively hinders necessary adaptation. After the agent learns to run in the backward direction for task 2 (Appendix 16), SI over-penalizes the large weight changes required to relearn running forward for task 3. The agent is effectively stuck with the backwards-running policy, resulting in negative rewards that are significantly worse than random initialization.
3. **Complexity of Pressing:** The high variance and lower transfer scores in the Pressing Sequence highlight the difficulty of transfer in contact-rich environments. The same is true for sub-optimal performances in Table 1. Unlike the kinematic reaching tasks, "pressing" involves complex dynamics and precise object manipulation. These lead to high sensitivity to random seeds across all baselines. Interestingly, SI ($\beta = 0.1$) shows superior forward transfer (Table 2). However, comes at the cost catastrophic forgetting (Figure 11b).
4. **DGR coming a close second:** DGR generally outperforms SI, serving as a close runner-up to HNET. Its limitation, however, lies in the quality of the generator. The continually trained generative model (VAE) produces noisy approximations of past transitions. While this prevents total forgetting, the noise leads to a sub-par reconstruction of past reward functions compared to HNET, which in contrast uses noise-free functional task embeddings.

10 Consolidated performance using Area Under Curve

In Table 3, we report the average over all points of evaluation (area under the curve) in addition to the final return-based evaluation as done in Table 1).

Domain	Metric	Oracle	Scratch	SI	DGR	HNET
Pendulum-v1	Continual training	-683.2 ± 22.7	$\mathbf{-780.0\pm43.0}$	-2157.3 ± 156.0	-850.6 ± 53.2	-864.8 ± 65.3
	Catastrophic forgetting	-681.4 ± 25.9	-3263.3 ± 70.2	-2906.6 ± 114.5	-1291.7 ± 157.0	$\mathbf{-830.4\pm63.2}$
Reach-v2	Continual training	4605.6 ± 14.6	3815.7 ± 68.5	3081.7 ± 232.5	$\mathbf{4862.9\pm18.8}$	4181.8 ± 54.5
	Catastrophic forgetting	4533.8 ± 27.4	1656.7 ± 112.0	1492.9 ± 157.9	2591.3 ± 241.5	$\mathbf{3924.2\pm257.1}$
Half Cheetah	Continual training	-210.5 ± 1.4	$\mathbf{-889.4\pm71.4}$	-1267.0 ± 238.8	-1035.0 ± 202.1	-897.5 ± 113.6
	Catastrophic forgetting	-254.7 ± 3.0	-1765.0 ± 101.7	-2045.0 ± 129.8	-1362.6 ± 243.5	$\mathbf{-1041.9\pm181.0}$
Reaching Sequence	Continual training	3831.6 ± 97.1	3025.7 ± 150.6	3016.1 ± 322.1	3128.7 ± 165.4	$\mathbf{3175.0\pm157.9}$
	Catastrophic forgetting	3718.2 ± 75.4	1468.7 ± 331.1	1496.6 ± 553.0	1973.5 ± 543.4	$\mathbf{2839.1\pm543.7}$
Pressing Sequence	Continual training	3201.8 ± 170.5	2326.4 ± 197.0	2316.1 ± 458.9	$\mathbf{2487.2\pm255.0}$	2428.7 ± 227.2
	Catastrophic forgetting	3050.7 ± 183.4	521.8 ± 412.5	1416.1 ± 477.2	527.6 ± 251.5	$\mathbf{1802.5\pm651.8}$

Table 3: Comparison of continual training and forgetting performance in terms of the consolidated metric across domains. Each entry reports mean $\pm$ confidence interval (95%) over 10 runs. Continual training performance measures the average return across all evaluation points, i.e., area under the curve (averaged over all tasks) during reward training, while catastrophic forgetting measures the average retained performance when using remembered reward function on previous tasks. *Higher is better*. For HNET, SI, and DGR, we report the results using hyperparameters that best balance the two metrics of learning new knowledge and preserving past knowledge. The objective is to achieve this balance (**RQ1**). Note: Oracle and Scratch are calibration baselines. The best mean performance (excluding Oracle) is in **bold**.

11 Statistical Significance Tests

To evaluate the statistical significance of the observed stability improvements, we conduct Welch's t-tests comparing the stability scores of HNET against those of each baseline across the experimental domains. The resulting p-values are summarized in Table 4. In the HalfCheetah and Reaching sequences, HNET demonstrates a statistically significant improvement ($p < 0.05$) over all baselines.

For the more challenging Pressing sequence, while HNET remains significantly better than the majority of baselines, the difference compared to high-regularization SI variants ($\beta \in \{1.0, 10.0\}$) is not statistically significant. This lack of significance in the Pressing domain likely stems from the high variance inherent in contact-heavy manipulation tasks, where performance distributions across seeds frequently overlap. Additionally, p-values for the training-performance experiments consistently exceeded the 0.05 threshold and were thus omitted from the table. This result is **expected**, as both HNET and the baseline methods often achieve performance levels near the oracle upper bound on a task during its active training phase and the resultant distributions are comparable.

For the results on Pendulum and Reach-v2, we observe that in many cases, the confidence intervals of HNET do not overlap with the confidence intervals of other baselines (Figures 5 and 2). This indicates that the difference between HNET and baselines in these two environments is statistically significant.

Domain	**Scratch**	**Synaptic Intelligence (SI)**			**Deep Gen. Replay (DGR)**	
		$\beta = 0.1$	$\beta = 1.0$	$\beta = 10.0$	$\lambda = 1.0$	$\lambda = 5.0$
HalfCheetah	1.839×10^{-8}	6.793×10^{-8}	2.593×10^{-9}	2.618×10^{-10}	0.0138	4.247×10^{-5}
Reaching Seq.	1.015×10^{-5}	3.647×10^{-5}	9.377×10^{-5}	1.578×10^{-5}	0.0002	0.0010
Pressing Seq.	4.438×10^{-5}	0.0002	0.1255	0.7230	5.427×10^{-5}	2.407×10^{-5}

Table 4: **Statistical Significance of Stability Improvements.** We report p-values from Welch's t-tests comparing the final **stability** scores (as also reported in Table 1) of HNET against each baseline across three domains. Values of $p < 0.05$ indicate statistically significant improvements by HNET. Values are rounded to four significant figures or scientific notation.

12 Need for Regularizer

An important hypothesis in HNET is that the regularizer $\mathcal{L}_{\text{reg}}$ (Equation 4) is necessary to anchor the hypernetwork's output manifold for previous task embeddings. To verify this, we conducted an ablation by setting $\beta_{reg} = 0$, effectively reducing our method to a standard multi-task training setup without explicit regularization constraints. We performed two sets of experiments here:

- We only set $\beta_{\text{reg}} = 0$ for our method but keep everything else unchanged (Appendix 12.1).
- We experiment with the original PEBBLE (without hypernetwork). The reward network is directly optimized sequentially with no resets at each task boundary. Note, that this is different from Scratch that resets networks at task boundaries (Appendix 12.2).

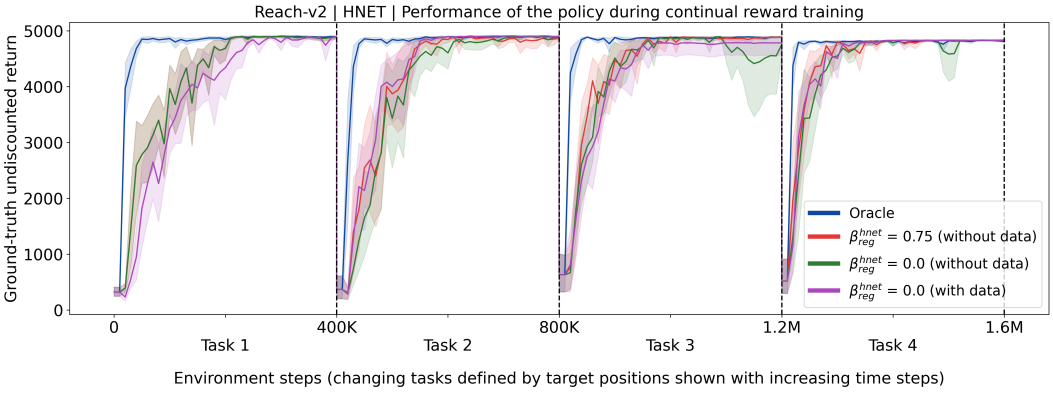
12.1 Ablation with $\beta_{\text{reg}} = 0$ in HNET

When setting $\beta_{\text{reg}} = 0$ in this ablation, the final policy performance during continual reward training is near-optimal for all the tasks. Hence, we only focus on the results for policy learning on the remembered reward after completing $N = 4$ tasks in Reach-v2 (Figure 13b).

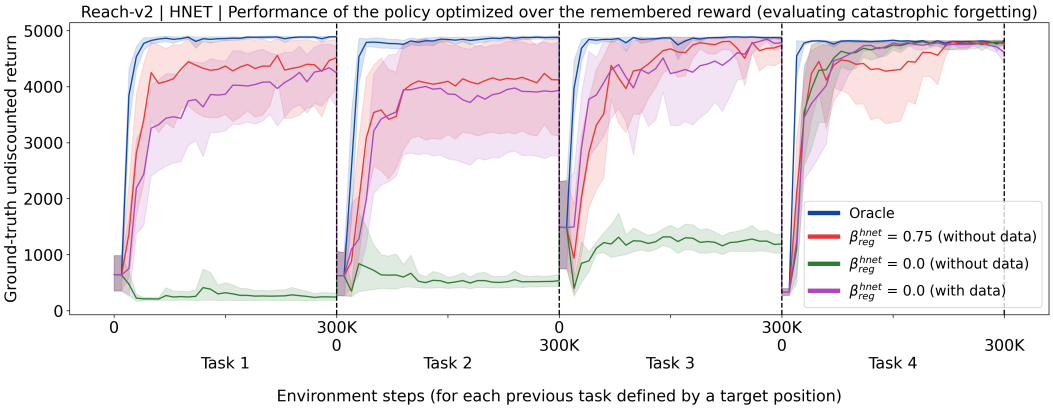
Figure 13b shows the extent of forgetting in HNET with and without the regularizer. Recall that the regularizer is weighted by the coefficient β_{reg}, which is non-zero for our main HNET experiments. When $\beta_{\text{reg}} = 0$, it is equivalent to switching off the regularizer. Clearly, HNET with $\beta_{\text{reg}} = 0$ forgets all tasks except the most recent one. This tells us that the regularizer is crucial for good HNET performance.

Further, we experiment with another HNET variant in which we keep the regularizer switched off; however, we give the agent access to past data. The agent using hypernetwork (with embeddings) resamples and retrains on past data to remember previous tasks. Naturally, we observe good performance across all tasks with this variation. Although storing past data is not allowed in CPRL, visualizing this result amplifies the usefulness of the regularizer. The regularizer enables achieving a similar level of performance as when storing and retraining on past data without storing any data.

A side note worth mentioning is that we had to reduce the hypernetwork training stepsize (by five times) with the HNET variant that uses past data. Not doing so results in slightly worse performance, especially on the first two tasks. This hints that the hypernetwork training procedure might be more unstable when using past data without the regularizer than when not doing so. It is possible that apart from balancing learning new and remembering old tasks, the regularizer also contributes to training stability. Exploring this direction might be an interesting future work for research on hypernetwork training dynamics.



(a) Performance of the policy during continual learning of reward functions (tasks) in Reach-v2. The curves indicate whether close-to-oracle performance can be achieved on each sequential task.

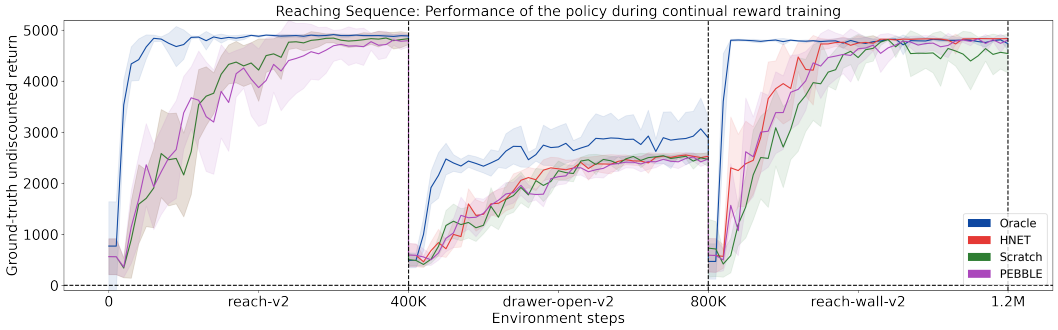


(b) Performance of the policy (on each previous task) obtained by optimizing over the (learned) rewards $r^N_{\psi_k}$ $(1 \leq k \leq N)$ in Reach-v2. The reward has been trained continually for $N = 4$ tasks. The policy performance curves (indirectly) indicate how well the memory of previous tasks' reward functions is retained. The x-axis is discontinuous, representing policy optimization on N independent tasks without reward learning.

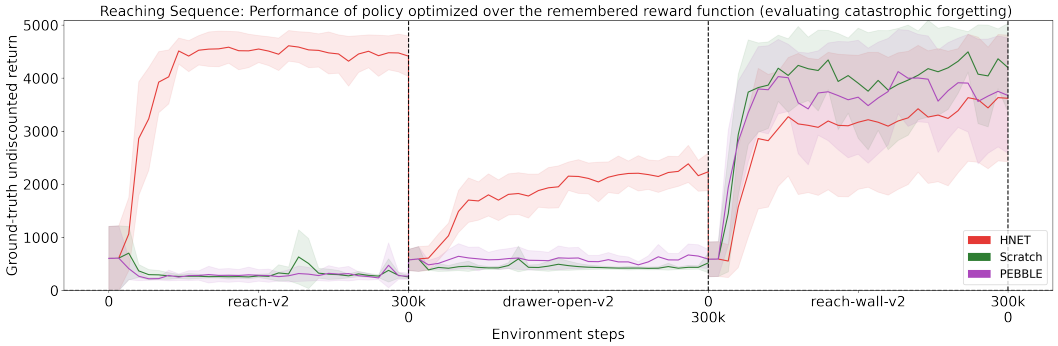
Figure 13: Ablation study over the regularizer for HNET in Reach-v2. These results show that the regularizer is significant for good HNET performance in Reach-v2.

12.2 Ablation using PEBBLE (without hypernetwork)

To conclusion this section, the unregularized models exhibits immediate catastrophic forgetting. The final learned reward performs disastrously on past tasks (Figs 13, 14 in Appendix 12). This



(a) Performance of the policy during continual learning of reward functions (tasks) in the Reaching sequence. The curves indicate whether close-to-oracle performance can be achieved on each sequential task.



(b) Performance of the policy (on each previous task) obtained by optimizing over the (learned) rewards $r^N_{\psi_k}$ $(1 \leq k \leq N)$ in the Reaching sequence. The reward has been trained continually for $N = 3$ tasks. The policy performance curves (indirectly) indicate how well the memory of previous tasks' reward functions is retained. The x-axis is discontinuous, representing policy optimization on N independent tasks without reward learning.

Figure 14: Ablation study over the regularizer for HNET in the Reaching sequence. Here, we run PEBBLE on each task without resetting any of its component networks. These results show that PEBBLE alone without any regularizer causes overfitting to the last task, resulting in catastrophic forgetting.

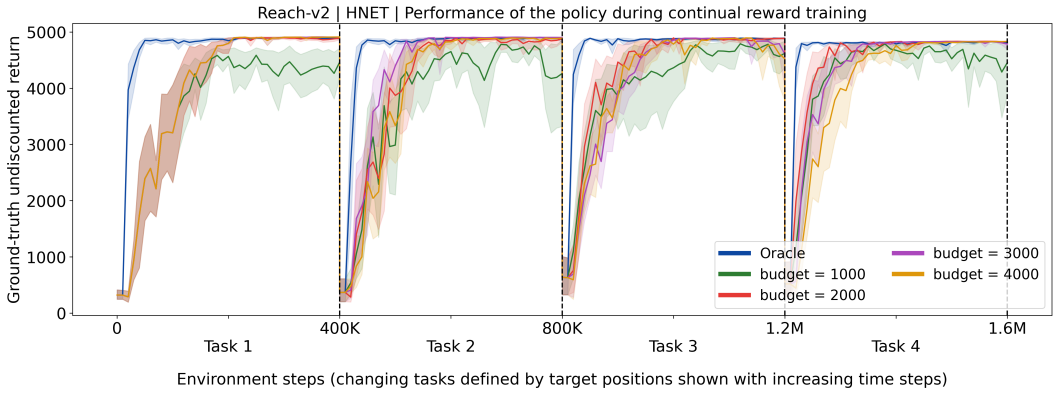
confirms that the distinct embeddings e^i alone are insufficient to isolate task knowledge and that explicit output regularization is critical for CPRL with HNET.

13 Reward Overfitting

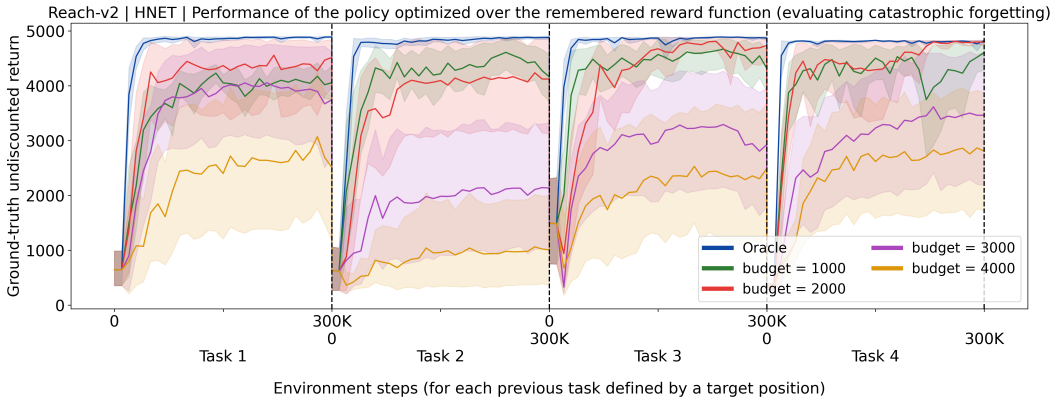
We observe an interesting phenomenon exhibited by our algorithm when the number of teacher queries (budget) was increased well beyond the necessary budget ($3000, 4000$ in Figure 15). The performance of the final learned reward in training a fresh SAC policy (Figure 15b) *drops* as the budget increases when tested on the Meta-world Reach-v2 domain (Fig 15). This is counter-intuitive, but upon closer examination, we found that the learned reward function's alignment (τ_{tac}) with r^{gt} on trajectories with low-returns was particularly poor. We hypothesize that this is due to reward overfitting. The agent learns a near-optimal policy within a certain budget (here 2000).

As a result, the remaining preference queries for budgets 3000 and 4000 are collected using a near-optimal policy, which increases the proportion of such queries in the buffer. Hence, there is a higher probability that the remaining updates to the hypernetwork are done using queries sampled from a near-optimal policy rather than sub-optimal. Doing so overfits the hypernetwork (that outputs the reward parameters) to the trajectory spaces corresponding to high returns. As a result, the reward gets misaligned on sub-optimal spaces and becomes worse at distinguishing between lower and higher-return trajectories. When such a reward is used to optimize a policy from scratch, as done during forgetting evaluation, it is unable to guide the policy learning in the beginning as the initial policy is sub-optimal. This causes the policy to never navigate to high-return trajectory spaces as it is misguided from the beginning itself, resulting in the learning curves observed in Figure 15b.

However, this phenomenon was not observed with the HalfCheetah environment. This warrants further research and analysis, which we leave to future work.



(a) Performance of the policy during continual learning of reward functions (tasks) in Reach-v2. The curves indicate whether close-to-oracle performance can be achieved on each sequential task.



(b) Performance of the policy (on each previous task) obtained by optimizing over the (learned) rewards $r^N_{\psi_k}$ $(1 \leq k \leq N)$ in Reach-v2. The reward has been trained continually for $N = 4$ tasks. The policy performance curves (indirectly) indicate how well the memory of previous tasks' reward functions is retained. The x-axis is discontinuous, representing policy optimization on N independent tasks without reward learning.

Figure 15: HNET behavior with different per-task budgets in Reach-v2 in terms of performance during continual reward training (a) and forgetting evaluation (b). Note that (b) involves policy learning from scratch on the remembered reward function. This result is useful for showing the difference in the learning performance of (a) and (b) with HNET $(3000, 4000)$.

14 Forward Transfer

Metric Definition : Wołczyk et al. (2021) propose the Continual World benchmark. They propose a metric for Forward Transfer (FT) in terms of how much the learning of a new task is accelerated by the knowledge acquired from previous tasks. It is calculated as the area under the learning curve (AUC) of the continual learning method relative to learning tasks from scratch, normalized to the range $[0, 1]$:

$$FT_i = \frac{AUC_i - AUC_{ref}}{1 - AUC_{ref}},$$

where AUC_i is the area under the training curve for task i, and AUC_{ref} is the area for a baseline trained in isolation. A positive FT indicates that past knowledge facilitated faster learning (positive transfer), while a negative FT implies that previous priors hindered performance (interference/negative transfer).

Calculation and The Importance of Task 3 : Our implementation calculates this metric in three steps:

- Global Normalization: All training curves are first normalized to the $[0, 1]$ interval using the global minimum and maximum returns observed across all runs. This ensures the AUC calculation is scale-invariant and bounded.
- AUC Computation: We compute the trapezoidal integral of the normalized return over the training steps for each task.
- Relative Performance: Finally, we compare the method's AUC against the Scratch baseline using the formula above.

In our "sandwich" sequences (Task $1 \rightarrow$ Task $2 \rightarrow$ Task 3, where $T_1 \approx T_3$), FT on Task 3 is the critical metric. Because T_3 is structurally identical or highly similar to T_1, an ideal agent should exhibit high positive transfer, recalling the skills learned in T_1. However, the intermediate Task 2 causes interference.

15 More details of Algorithms

15.1 Algorithm for training HNET in CPRL

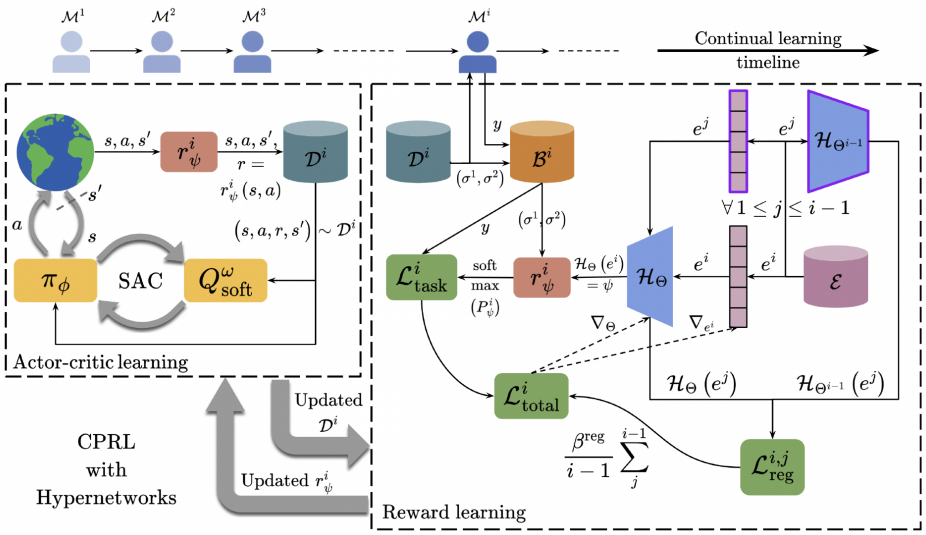


Figure 16: Workflow of CPRL with hypernetworks in Algorithm 1. The two sub-processes of reward learning (via hypernetworks) and actor-critic learning (via SAC) alternate as shown. The updated reward parameters from the reward learning phase are used for SAC updates. The SAC procedure adds transitions to the (task) replay buffer, from which segment-pairs are sampled to obtain feedback and update the hypernetwork and the embedding.

15.2 Algorithm for Forgetting Evaluation

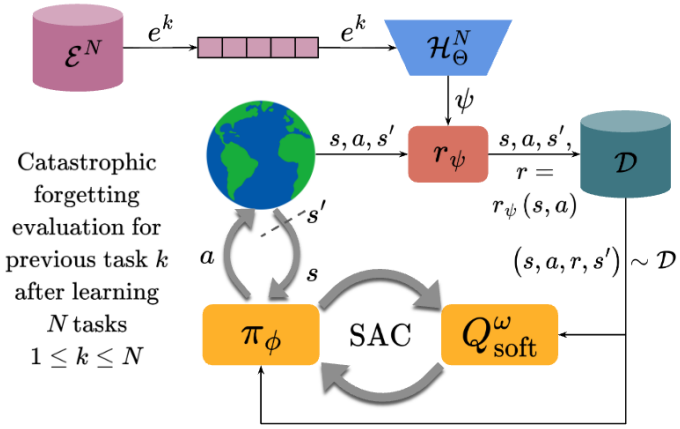


Figure 17: Catastrophic forgetting evaluation with hypernetworks in Algorithm 2 after experiencing $N > 1$ tasks. The embedding buffer and the hypernetwork (after N tasks) are used to generate the reward network parameters for a previous task $1 \leq k \leq N$. The reward is then used for actor-critic learning as per SAC. The notation used follows the definition from the previous sections.

Algorithm 2 Catastrophic forgetting evaluation with hypernetworks

Input: Hypernetwork $\mathcal{H}_\Theta^N$ and task embedding buffer $\mathcal{E}^N$ after learning all $N > 1$ tasks
Input: Task number (identity) of a previous task $1 \leq k \leq N$, minibatch size B
Initialize actor $\pi_\phi(a|s)$, critic $Q_{soft}^{\omega}(s,a)$, target critic $Q_{soft}^{\bar{\omega}}(s,a)$, temperature α
Initialize replay buffer $\mathcal{D} = \emptyset$
Initialize reward network r_ψ for task k with parameters $\psi = \mathcal{H}_\Theta^N(e^k)$, where $e^k \in \mathcal{E}^N$
Collect samples randomly in the environment: $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_d, a_d, r_\psi(s_d, a_d), s_{d+1})\}_d$
for each iteration **do**
  for each environment step t **do**
    Sample action $a_t \sim \pi_\phi(\cdot|s_t)$ and the next state $s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)$
    Compute the reward $r_{t+1} = r_\psi(s_t, a_t)$
    Store transitions $\mathcal{D} \leftarrow \mathcal{D} \cup \{(s_t, a_t, r_{t+1}, s_{t+1})\}$
  end for
  // Actor-Critic learning
  for each gradient step **do**
    Sample minibatch $\{(s_j, a_j, r_{j+1}, s_{j+1})\}_{j=1}^{B} \sim \mathcal{D}$
    Update critic, actor and temperature.
    Update target critic parameters $\bar{\omega}$.
  end for
end for

16 Implementation

16.1 Environement Details

Table 5: Environments and Task Sequences for Continual Preference-based RL

Env Name	Goal Changed	Actual Tasks (Sequence)
Pendulum-v1	Target Angle (Ω_{tgt})	$[120°, 90°, -30°, 60°, 0°, -90°, -120°, 150°, -60°]$
Reach-v2	Target Position (x, y, z)	1. $(0.18, 0.5, 0.05)$ 2. $(-0.16, 0.7, 0.29)$ 3. $(0.05, 0.8, 0.09)$ 4. $(-0.09, 0.88, 0.19)$
HalfCheetah-v4	Goal Velocity	$[2.0, -2.0, 1.0]$

HalfCheetah's goal velocities have been chosen using the same logic as mentioned in the main paper for the two MetaWorld Sequences. We wanted the first and last task to be similar, while the second quite different.

Note that the Reaching and Pressing Sequences do not have any set goals. They are just different environments in sequence.

16.2 Hyperparameters

Table 6: SAC Hyperparameters across Environments

Parameter	**Pendulum**	**Reach-v2**	**HalfCheetah**	**MW-Sequences**
Optimizer	Adam	Adam	Adam	Adam
Discount Factor (γ)	0.99	0.99	0.99	0.99
Batch Size	64	256	256	256
Actor/Critic/α LR	10^{-3}	7×10^{-4}	7×10^{-4}	7×10^{-4}
Target Smooth (τ)	0.005	0.005	0.005	0.005
Hidden Dim	16	64	64	64
Hidden Layers	2	3	3	3
Replay Buffer	∞	∞	∞	∞

Table 7: Preference Learning

Parameter	**Pendulum**	**Reach-v2**	**HalfCheetah**	**MW-Sequences**
Segment Length	25	50	50	50
Feedback Budget	200	2,000	2,000	2,500
Teacher Feedback Freq (K)	5,000	5,000	5,000	5,000
Reward LR	10^{-3}	5×10^{-4}	5×10^{-4}	5×10^{-5}
Reward Batch	64	128	128	128
Queries per Feedback Session	50	50	50	50
Sampling Technique	Uniform	Uniform	Uniform	Uniform

Table 8: Hypernetwork (HNET) Specific Hyperparameters

Parameter	**Pendulum**	**Reach-v2**	**HalfCheetah**	**MW-Sequences**
Embed Dim ($dim(e)$)	10	32	32	32
HNET Hidden Dim	16	32	32	32
HNET Layers	2	3	3	3
Stepsize	10^{-3}	5×10^{-4}	5×10^{-4}	7×10^{-5}
Reg Coeff (β_{reg})	0.75	0.75	0.3	0.7

16.3 Trajectory Alignment Coefficient: Kendall's Tau

This metric quantifies the correlation between the ranking of trajectory segments induced by the learned reward model r_ψ and the ground-truth preference ranking. It is computed using Kendall's rank correlation coefficient (Kendall's τ-b):

$$\tau_{tac} = \frac{P - Q}{\sqrt{(P + Q + T_L)(P + Q + T_G)}} \in [-1, 1] \tag{5}$$

where P is the number of concordant pairs (rankings agree), Q is the number of discordant pairs, and T_L, T_G account for ties in the learned and ground-truth rankings, respectively. A score of 1 indicates a perfect alignment with the teacher's preferences, -1 implies complete opposite alignment, and 0 is as good as a random ordering.