Applied Intelligence (2022) 52:7459–7479
https://doi.org/10.1007/s10489-021-02726-3

# Hierarchical learning from human preferences and curiosity

Nicolas Bougie[1,2] · Ryutaro Ichise[1,2]

Accepted: 27 July 2021 / Published online: 28 September 2021

**Abstract**
Recent success in scaling deep reinforcement algorithms (DRL) to complex problems has been driven by well-designed extrinsic rewards, which limits their applicability to many real-world tasks where rewards are naturally extremely sparse. One solution to this problem is to introduce human guidance to drive the agent's learning. Although low-level demonstrations is a promising approach, it was shown that such guidance may be difficult for experts to demonstrate since some tasks require a large amount of high-quality demonstrations. In this work, we explore human guidance in the form of high-level preferences between sub-goals, leading to drastic reductions in both human effort and cost of exploration. We design a novel hierarchical reinforcement learning method that introduces non-expert human preferences at the high-level, and curiosity to drastically speed up the convergence of subpolicies to reach any sub-goals. We further propose a strategy based on curiosity to automatically discover sub-goals. We evaluate the proposed method on 2D navigation tasks, robotic control tasks, and image-based video games (Atari 2600), which have high-dimensional observations, sparse rewards, and complex state dynamics. The experimental results show that the proposed method can learn significantly faster than traditional hierarchical RL methods and drastically reduces the amount of human effort required over standard imitation learning approaches.

**Keywords** Hierarchical reinforcement learning · Preference-based learning · Curiosity · Human guidance

## 1 Introduction

In reinforcement learning (RL), the objective of the agent is set by an extrinsic reward function. However, such reward functions are often poorly-defined, sparse, or delayed, which may severely limit the applicability of reinforcement learning methods. Moreover, RL algorithms often require a large number of interactions to reach decent performance, which can be intractable in real-world settings. A strategy to escape these pitfalls is hierarchically reinforcement learning (HRL) [50] that decomposes the overall task into easier short-term sub-tasks. However, it may be slow to learn subpolicies and it remains challenging to order subpolicies in the absence of dense task rewards. Besides, HRL usually requires to manually define a set of sub-goals.

✉ Nicolas Bougie
nicolas-bougie@nii.ac.jp

Ryutaro Ichise
ichise@nii.ac.jp

[1] The Graduate University for Advanced Studies (Sokendai), Tokyo, Japan

[2] National Institute of Informatics, Tokyo, Japan

Another successful class of methods for dealing with such problems is imitation learning (IL) [43] in which the agent learns by observing and possibly querying an expert [14]. Nevertheless, these approaches are not directly applicable to behaviors that are difficult for humans to demonstrate such as robot control or temporally-extended tasks, and assume that the human demonstrator has some familiarity with the task. A solution is to leverage other types of human guidance such as "preferences" [16] or "weak feedback" [66]. These types of human guidance were shown to be easier to demonstrate for humans and reduce the amount of human involvement. For instance, preference-based learning has been applied to game-playing as well as robot control [16, 28]. However, we argue that providing low-level preferences between pairs of trajectory segments may be an inefficient way of soliciting humans, and could be expensive and/or subject to error. A different paradigm provides an intrinsic exploration bonus (i.e. curiosity) to the agent. For example, count-based exploration [51] keeps visit counts for states and favors the exploration of states rarely visited. Curiosity can also be measured as the error in predicting the consequences of the agent's actions on the environment [42], a prediction task where the problem is a deterministic function of its inputs [9, 13], or explicitly promote in-depth exploration [10]. Prior work have demonstrated that curiosity is a sufficient exploration signal to improve control and execution in agents (e.g. learning to reach a sub-goal in HRL), however, learning a task from scratch can still require a prohibitively time-consuming amount of exploration of the state-action space in order to find a good policy. Namely, reasoning and planning based on common sense priors [24] is beyond the reach of agents trained with no prior assumptions about the domain. In order to expand the availability of DRL, it is necessary to combine the strengths of both of these techniques. In other words, human preferences is a good source of guidance for high-level decision-making such as planning/reasoning, while curiosity is a good source of endogenous motivation for exploring the consequences of low-level actions on the environment (control and execution), like how to pass an obstacle or whether to interact with a particular object.

In this paper, we propose a hierarchical reinforcement learning algorithm. At a high-level, a meta-controller policy is trained to select sub-goals given (non-expert) human high-level preferences between pairs of sub-goals as a feedback signal, leading to dramatic reductions in both human workload and degree of familiarity necessary to provide feedback. Besides, human preferences provide prior assumptions about the domain to the agent, avoiding learning from scratch and enabling common sense reasoning. We decide how to query preferences based on an approximation to the confidence in the action selection. At a low-level, we introduce curiosity to drive the learning of subpolicies (i.e. low-level control and execution). Such an approach is motivated by learning in animals, they utilize all possible intermediate learning signals (e.g. curiosity, preferences) to solve challenging tasks. We further contribute a strategy to automatically discover sub-goals that relies on the agent's curiosity, alleviating the need for an expert to define sub-goals. As an intuition, we found that manually created sub-goals are generally associated with large spikes in the curiosity, which indicate meaningful events. This strategy promotes active cooperation between levels of the policy by communicating potential novel meaningful sub-goals throughout the training process. Our experiments take place in three domains: 2D navigation tasks in Minigrid, robotics tasks in the physics simulator MuJoCo, and Atari games in the Arcade Learning Environment (ALE). We show that a small amount of feedback from a non-expert human suffices to rapidly learn both standard RL tasks and novel hard-to-specify behaviors such as playing Montezuma's Revenge or dexterous manipulation with robotic arms, notorious for their extremely sparse rewards and complexity.

The key contributions of this article are as follows:

- We present a two-level hierarchical algorithm that introduces human high-level preferences between pairs of sub-goals at the high-level to drastically reduce human workload, while being more intuitive for humans and enabling non-expert feedback.
- We introduce a technique to actively request very few preferences - in states where the agent's confidence is low.
- We propose to use curiosity at the low-level to drive exploration and reduce the amount of interactions to learn subpolicies (i.e. low-level policies). Next, we derive a method to automatically discover sub-goals through the idea of *curiosity-driven sub-goal discovery*.

## 2 Background

Our method builds on top of reinforcement learning and hierarchical learning. We briefly introduce them in this section.

### 2.1 Reinforcement learning

Reinforcement learning [52] consists of an agent learning a policy $\pi$ by interacting with an environment. At each time-step the agent receives an observation $s_t$ and chooses an action $a_t$. The agent gets a feedback from the environment called reward, $r_t$. Given this reward and the current observation, the agent can update its policy to improve the future expected rewards. Given a discount factor $\gamma$, the future discounted rewards, called return $\mathcal{R}_t$, is defined as $\mathcal{R}_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$, where $T$ is the time-step at which the epoch terminates. The agent learns to select the action with the maximum return $\mathcal{R}_t$ achievable for a given observation [53]. We can define the action value $Q^{\pi}(s, a)$ at a time $t$ as the expected reward for selecting an action $a$ for a given state $s_t$ and following a policy $\pi$: $Q(s, a) = \mathbb{E}[R_t \mid s_t = s, a]$. The policy being learned is called the target policy, and the policy used to generate exploration behavior is called the behavior policy.

### 2.2 Hierarchical learning

In this paper, we consider a hierarchical RL agent with a two-level hierarchy (Fig. 1). The meta-controller policy (also called master policy or high-level policy) chooses a sub-goal that a low-level policy tries to reach. For instance, a sub-goal could be "reach the door" and the corresponding subpolicy "go left" → "jump" → "go down" → "use key". Each sub-goal corresponds to one subpolicy that can be executed by the low-level component. We denote a sub-goal as $g \in \mathcal{G}$ and a low-level action is denoted by $a \in \mathcal{A}$. The high-level policy iteratively selects a sub-goal $g$ which is then executed by the corresponding subpolicy until completion or failure. To choose this sub-goal, the

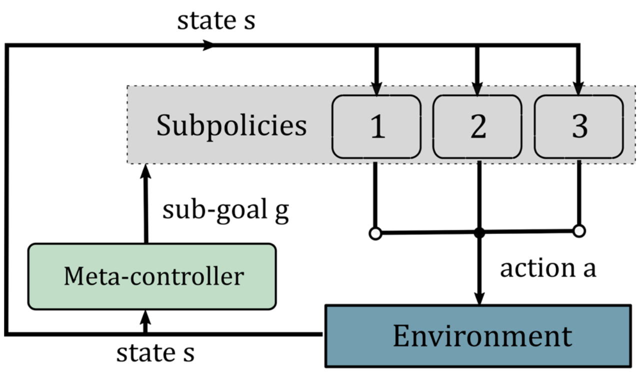


**Fig. 1** Hierarchical reinforcement learning

high-level policy takes as input the current state $s \in \mathcal{S}$. This problem can be formalized as simultaneously learning the high-level policy $\mu : \mathcal{S} \rightarrow \mathcal{G}$ and a set of low-level policies $\pi_g : \mathcal{S} \rightarrow \mathcal{A}$. Low-level policies receive a positive reward when the sub-goal being pursued is achieved. On the other hand, the meta-controller receives an extrinsic reward provided by the environment, which indicates whether the agent is improving at solving the overall task.

## 3 Related work

Methods for providing external supervision largely divide into two categories: imitation learning and learning from interactive human feedback. We briefly introduce these techniques in this section. Next, we discuss other sample-efficient hierarchical reinforcement learning methods and curiosity-driven learning.

### 3.1 Imitation learning

DQfD [25] pre-trains a Q-learning agent on the expert demonstration data. This idea was extended to handle continuous action spaces such as in robotic tasks [58], as well as to actor-critic architectures [67]. In order to allow the agent to outperform the expert, a recent follow-up [39] introduces an expert loss in DDPG [35] and proposes to filter suboptimal demonstrations based on the Q-values. To reduce the amount of necessary demonstrations, a solution is to represent a policy as a set of Gaussian mixture models [14] and select needed demonstrations based on the model's uncertainty. In this work, we introduce a simple strategy to model the agent's confidence that relies on *dropout*. One advantage of using dropout is that it allows the predictor network to “smooth out” much of the noise in the data, making the preference policy more robust to noise in the demonstration data. Besides, this technique is particularly effective in the low-data regime to improve generalization of demonstrated preferences. In a different spirit, AlphaGo [49] trains a policy network to classify positions according to expert moves. A way of dealing with sparse rewards consists in introducing a curious replay mechanism and demonstrations [69]. In recent years, an emerging strategy combines generative adversarial networks and reinforcement learning (GAIL) [26]. However, GAIL was shown to suffer from the *drift prediction error* and instability during training. Another method [19] constructs a goal-conditioned policy to visit similar states as the expert. A different form of imitation learning is inverse reinforcement learning (IRL) [40] that uses demonstrated trajectories to extract a reward function. IRL has been applied to several domains including navigation [4, 68], and autonomous flight [1]. However, in many cases it is impractical to demonstrate long-term tasks and IRL algorithms assume that the observed behavior is optimal. Overall, it is not clear how to scale imitation learning to much more complex behaviors that are difficult to collect. Another limitation is how to learn when the agent's goal deviates from the demonstrated trajectories. This work differs by leveraging high-level guidance - human preferences between pairs of sub-goals, alleviating the need to collect low-level trajectories (e.g. demonstrations) while drastically reducing human effort.

### 3.2 Interactive human feedback

Most methods that focus on learning from interactive human feedback [16, 28, 37, 63, 64] query the human to drive learning. Especially, it is possible to query trajectory preferences [16], which can be combined with fixed demonstrations [28]. In contrast, the depicted method introduces the idea of high-level preferences, and combines human feedback with curiosity that yields better-than-expert performance by guiding the agent in the quest of knowledge beyond the expert's knowledge. Besides, our approach builds upon hierarchical learning and introduces a novel technique to decide how to query. Another example, TAMER [62], trains the policy from feedback in high-dimensional state spaces. The learner may also receive feedback in the form of sequences of actions planned by a teacher [11]. Some authors [48] consider multiple demonstrators performing different tasks and the agent must actively select which one to request for advice. Another solution [45] is to block unsafe actions by training a module from expert feedback. However, it requires the expert to identify all unsafe situations by watching an agent play. We argue that a constant supervision seems impractical, so our work lets the agent identify querying opportunities so that the expert is not required to constantly monitor the agent. To deal with the problem of query selection, it is possible to select sufficiently different unqueried data [27]. In this work, we only request high-level feedback to the supervisor in states where the agent is unsure and struggles.

Moreover, the proposed form of human guidance does not necessarily require the human trainer to be an expert at performing the task since the proposed form of guidance only requires the human to judge outcomes. We further use the structure of the task and introduce curiosity to drive the agent's exploration of sub-tasks for which human feedback is not necessary, further reducing human effort.

### 3.3 Hierarchical reinforcement learning

One common strategy to solve temporally extended and/or complex tasks is to exploit the hierarchical structure of the task [54]. Several approaches assumed a set of pre-specified options - a temporally extended sequence of low-level actions, and planning / learning only occurs at the higher level [22, 54]. While earlier works used pre-specified options, a paper proposes a framework capable of learning both the options and the termination conditions of options [5]. Feudal learning is a multi-level hierarchical model where communication is made between *managers* and *workers* through goals and rewards [59]. However, it is tailored to a specific kind of problem. MAXQ is a hierarchical learning algorithm in which the hierarchy of a task is obtained by decomposing the Q-value of a state-action pair into the sum of two components. Contrary to *options*, the MAXQ framework introduces a real hierarchical decomposition of the task and it facilitates the reuse of sub-policies [18]. In recent years, to address efficiency, HIRO proposes to use off-policy experience for both higher and lower-level training [38]. This algorithm was then extended to jointly learn a hierarchy of policies. They make use of goal-conditioned policies that use the state space as the mechanism for breaking down a task into subtasks [34]. In this work, we propose to discover sub-goals based on the agent's curiosity. In addition, we address the problem of sample efficiency by introducing human feedback to massively accelerate the learning of high-level policies, and curiosity at the low level.

### 3.4 Sample-efficient hierarchical reinforcement learning

As mentioned above, many tasks are hierarchically structured, which entails that they can be decomposed and gradually solved. In order to improve sample efficiency, several work aim to introduce external or internal supervision. A closely related hierarchical RL work to ours is the approach named hg-DAgger [33] in which the agent can request high-level and low-level demonstrations, and a signal to indicate if the high-level sub-goal was chosen correctly. At each level, the learning task becomes a typical imitation learning problem. In this work, we primarily focus on preferences to enable non-expert feedback and expand the possible applications of HRL, including in tasks with behaviors difficult to demonstrate. Moreover, we overcome the need for expert-engineered sub-goals by introducing a method to automatically discover sub-goals during exploration. Humans can also provide policy sketches that are high-level sub-task labels [2]. A forward model has also been used as a measure of novelty to improve sample efficiency in actor-critic HRL [44]. Integrating temporal abstraction and intrinsic motivation has also been applied to game-playing [31]. That is, they use a two-level hierarchical model where the top-level function learns a policy over intrinsic goals, and the low-level learns a policy over atomic actions. On the other hand, our method is driven by a Random Network Distillation [13] error that gradually downmodulates states that become progressively familiar across the agent's learning, escaping from the known “Noisy-TV” issue inherent in curiosity (see Section 3.6). Furthermore, while curiosity is a powerful source of endogenous motivation for low-level control, we found human guidance to be much more effective for high-level control, such as planning. Thus, we propose a framework that can integrate human guidance and curiosity at different levels to leverage the strengths of both learning signals.

### 3.5 Self-generating goals

Developing machine learning systems that are capable to explore efficiently by self-generating goals is critical to solve complex tasks [32]. One strategy is to train a generative model and then sample increasingly difficult goals from the model [21]. Another approach relies on the learning process at different instants in order to select new goals [6]. It is also possible to sample goals from a goal space that has been learned by the agent [32]. This goal space represents the set of possible outcomes that can be produced by the agent. As mentioned earlier, our method works in the state space (e.g., images) in order to allow a human demonstrator to provide feedback. The idea of skill discovery is also closely related to this field of research. In DIAYN [20], a skill is a latent-conditioned policy that dictates the states that the agent visits. They propose to discover skills by maximizing an information theoretic objective using a maximum entropy policy. In this work, goals are particular states that are selected based on the agent's novelty progress. Our method is easy to implement and allows the discovery of new goals throughout the training process, which may be seen as a form of implicit curriculum.

### 3.6 Curiosity-driven exploration

Inspired by curious behavior in animals - observing something novel or surprising could be rewarded with a bonus, the use of intrinsic motivation has been developed to promote agents to learn about their environments even when extrinsic feedback is rarely provided. A line of work is to keep visit counts for states to favor exploration of rarely visited states [7, 36, 51]. To enable count-based exploration in continuous state spaces, a solution [41] is to train an observation density model to supply counts. Another strategy [55] is to map states to hash codes and count state visitations with a hash table. In this setting, the counts are used as exploration bonus to guide exploration. A famous algorithm, ICM [42], relies on predicting environment dynamics using an inverse or forward dynamic model. That is, they train a predictor for the embedding of the next observation, and reward the agent if the reality is significantly different from the prediction. However, one major issue of prior work that measure the inability of the agent to predict the future is the "Noisy-TV" issue [13, 46]. Agents tend to get attracted to transitions where the answer to the prediction problem is a stochastic function of the inputs, for instance, a noisy TV in the environment that displays new images in an unpredictable order. This is a source of stochasticity that causes every next state to be unpredictable. To deal with this issue, RND proposes an alternative solution to this undesirable stochasticity by defining an exploration reward using a prediction problem where the answer is a deterministic function of its inputs [13]. Please note that they do not consider a truly random TV, but instead a TV where the images keep changing in a random fashion. Since the prediction is deterministic for each state, after visiting enough states coming from stochastic regions of the environment, the prediction error will decrease in such regions. On the other hand, other work will continuously try to find a connection between the agent's action and the resulting state.

## 4 Method

The challenges of injecting expert feedback into DRL are two folds. First, in many cases it is impractical to use human policy as guidance because some of these tasks are too challenging for even humans to perform well. Second, providing low-level feedback can be time-consuming and significantly increase human workload.

The framework of hierarchical learning from human preferences (HhP) provides us a mechanism to mitigate these problems (Fig. 2). Our approach introduces (non-expert) human high-level feedback into hierarchical reinforcement learning via preferences over sub-goals. This provides a significant speedup in learning while requiring less human effort. Although a number of algorithms could in principle be used to learn the low-level subpolicies, they often require to manually engineer sub-goals and rely on large amounts of interactions. In contrast, our formulation introduces curiosity to enable self-discovery of sub-goals and speed up the learning of subpolicies.

In the following section, we first describe high-level components of our method and then low-level components.

### 4.1 High-level preferences

In this section, we describe high-level components of our algorithm. When learning from demonstrated trajectories, the policy is trained to clone a human (expert) demonstrator on the task. Nevertheless, to provide meaningful demonstrations, the demonstrator has to have some knowledge and familiarity with the current task. As a result, learning from direct demonstrations of trajectories or high-level actions (i.e. sub-goals) is significantly more costly than requesting human (non-expert) preferences, which only uses the ability to judge outcomes. Moreover, rather than relying on trajectory preferences (i.e. low-level preferences) we propose to query high-level preferences, decreasing the amount of feedback required by several orders of magnitude.

To this end, we assume a set of sub-goals $\mathcal{G}$ and access to a supervisor that can provide preferences over sub-goals. Our method maintains a meta-controller (also called high-level policy) $\mu : \mathcal{S} \rightarrow \mathcal{G}$ and a *predictor* network that estimates a reward function from the annotator's preferences $\hat{p} : \mathcal{S} \times \mathcal{G} \rightarrow \mathbb{R}$, each parametrized by deep neural networks. Our predictor network takes a state and a sub-goal as input and outputs an estimate of the corresponding reward. In

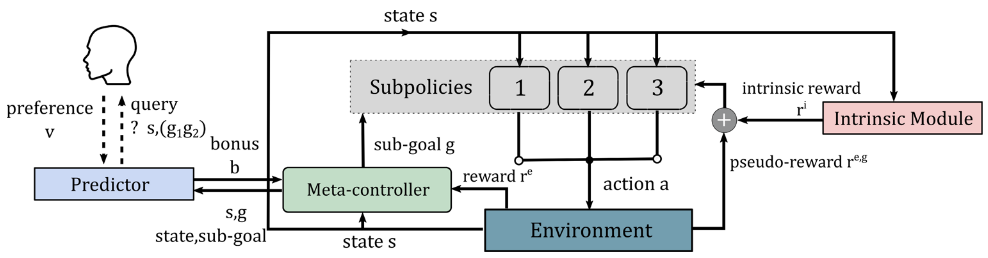


**Fig. 2** Hierarchical learning from human preferences and curiosity. The predictor network is trained based on high-level preferences provided by a demonstrator. The predictor network provides a bonus $b$ to the meta-controller for agreeing with human preferences. In addition to the pseudo-reward $r^{e,g}$, the subpolicies receive an intrinsic reward $r^i$

our settings, preferences are collected while the agent is training, during the experiment.

In detail, our algorithm works as follows (see Algorithm 1). For each selected sub-goal $g$ by the meta controller $\mu$ (line 11), the sub-policy $\pi_g$ selects and executes low-level actions until some termination condition is met (lines 15-21). Along with the pseudo reward (line 17), the agent receives an intrinsic reward (line 18). Upon termination of the sub-goal, another sub-goal is selected and the process continues until the end of the episode, where the involvement of the demonstrator begins. The expert is requested preferences (line 26) for the sub-goals where the predictor's confidence is low (line 25). The preferences are used to train the predictor network (line 27). Then, the agent inspects the last state trajectory and selects new goals based on its novelty-progress (line 29). Finally, the meta-controller is trained given preferences of the predictor network and extrinsic returns (lines 30-34). This process continues until the task is mastered. In the following section we describe the key components of our method: (1) training the predictor network to fit the high-level preferences collected from human so far, (2) optimizing the meta-controller, and (3) selecting new queries.

**Algorithm 1** Hierarchical learning from human preferences (HhP).

```
1:  Given:
      – replay buffer $R_H$, preference buffer $R$, low-level data buffers $D_g$, and RND buffer $\Omega$
      – meta-controller policy $\mu$, predictor network $\hat{p}$, subpolicies $\pi_g$, and RND model
      – predicates $done(s, g)$, $terminal(s, g)$, $alpha$, and $t_{qry}$
2:  Initialize $\mu$, $\hat{p}$, RND model
3:  Initialize $R_H = \{\}$, $R = \{\}$, and $D_g = \{\}$, $\Omega = \{\}$
4:  Initialize set of sub-goals $G = \{\}$
5:  Add a random sub-goal to $G$ and initialize the corresponding subpolicy $\pi_g$
6:  for m=0,...,M episodes do
7:      Receive initial state $s_0$ from the environment, $s_t \leftarrow s_0$
8:      $\varphi_H = \{\}$
9:      repeat
10:         $\varphi_L = \{\}$,
11:         Select a sub-goal $g \leftarrow \mu(s_t)$
12:         Calculate preference bonus $b_t \leftarrow \hat{p}(s_t, g|\theta^*)$
13:         Append $(s_t, g, b_t)$ to $\varphi_H$
14:         repeat
15:             Get action $a_t \leftarrow \pi_g(s_t)$
16:             Execute $a_t$ and observe next state $s_{t+1}$
17:             Compute pseudo-reward $r_t^{e,g}$ and intrinsic reward $r_t^i$
18:             Compute the total reward $r_t = r_t^{e,g} + \alpha \cdot r_t^i$
19:             Append $(s_t, a_t, s_{t+1}, g, r_t)$ to $\varphi_L$ and $D_g$
20:             Update $\pi_g$: a (stochastic) gradient descent step on a minibatch from $D_g$
21:             $s_t \leftarrow s_{t+1}$
22:         until $terminal(s_t, g)$
23:     until end of episode
24:     Calculate return for each sub-goal in $\varphi_H$
25:     Evaluate confidence for tuples in $\varphi_H$
26:     Query preference for low-confidence tuples and store preferences in $R$        ▷ Threshold $t_{qry}$
27:     Update $\hat{p}$ on minibatches from $R$                                        ▷ If a query was produced
28:     Add $\varphi_H$ to $R_H$
29:     Inspect $\varphi_L$ to select new sub-goals and add them to $G$                 ▷ Uses models in $\Omega$
30:     for j=0,...,$N_{opt}$-1 do
31:         Sample a minibatch $B$ from $R_H$
32:         Normalize the preference bonus contained in $B$
33:         Calculate rewards $r^e$
34:         Update meta-controller $\mu \leftarrow \text{TRAIN}(\mu, B, r^e)$
35:     Update the RND model and potentially replace models in $\Omega$
```

### 4.1.1 Training the predictor network

Our model maintains a predictor network $\hat{p}$ that mimics human high-level preferences. It takes as input the current state $s_t$ and the sub-goal $g$ recommended by the meta-controller, and outputs a reward that "criticizes" the meta-controller's recommendation, $b_t \in \mathbb{R}$.

In order to train the predictor model, the human overseer is given a visualization of a state $s$ and a pair of two sub-goals $(g_1, g_2)$, in the form of images. The human then indicates which sub-goal is favoured, that the two sub-goals are equally good, or that the overseer is unable to compare the two sub-goals. We record tuples $(s, g_1, g_2, v)$ in a database $\mathcal{R}$, where $s$ is a state, $g_1$ and $g_2$ are two distinct sub-goals, and $v$ is the score given by the demonstrator. Write $g_1 \succ g_2$ to indicate that the human preferred sub-goal $g_1$ to sub-goal $g_2$, and $g_1 \not\succ g_2$ to indicate that the two sub-goals are equally good. In our implementation we use labels for encoding preferences (i.e. score $v$): [1, 0, 0] denotes $g_1 \succ g_2$, [0, 1, 0] denotes $g_2 \succ g_1$, and [0, 0, 1] denotes $g_1 \not\succ g_2$.

Assuming the data-set $\mathcal{R}$ of tuples, $\mathcal{R} = \{(s, g_1, g_2, v), ...\}$, we train the predictor network $\hat{p}$ to minimize the cross-entropy loss, $\mathcal{L}$, between these predictions and the actual judgment labels. Since we discard tuples that are incomparable, the loss function can be written as follows:

$$\mathcal{L}(\hat{p}, \theta^*, \mathcal{R}) = -\frac{1}{|\mathcal{R}|} \sum_{(s,g_1,g_2,v)\in\mathcal{R}} \log(P[g_1 \succ g_2])v_0 + \log(P[g_2 \succ g_1])v_1 \quad (1)$$

where $\theta^*$ is the set of parameters of the predictor network, $v_i$ corresponds to the $i$'th element of one-hot encoded label of the sample $(s, g_1, g_2)$, $(g_1, g_2)$ is a pair of sub-goals, and $P$ is the probability of preferring a sub-goal. We calculate this probability via:

$$P[g_1 \succ g_2] = \frac{e^{\hat{p}(s,g_1|\theta^*)}}{e^{\hat{p}(s,g_1|\theta^*)} + e^{\hat{p}(s,g_2|\theta^*)}} \quad (2)$$

Therefore, the overall loss can be re-written as:

$$\mathcal{L}(\hat{p}, \theta^*, \mathcal{R}) = -\frac{1}{|\mathcal{R}|} \sum_{(s,g_1,g_2,v)\in\mathcal{R}} \log\left[\frac{e^{\hat{p}(s,g_1|\theta^*)}}{(e^{\hat{p}(s,g_1|\theta^*)} + e^{\hat{p}(s,g_2|\theta^*)})}\right]v_0 + \log\left[\frac{e^{\hat{p}(s,g_2|\theta^*)}}{(e^{\hat{p}(s,g_2|\theta^*)} + e^{\hat{p}(s,g_1|\theta^*)})}\right]v_1$$

This loss function is a specialization of the Bradley-Terry model [29] for estimating score functions from pairwise preferences. Similarly to the low-level preference model [16], we resort to indirect training of this model via preferences expressed by the teacher. However, here, rather than working on low-level segment trajectories, our method operates on sub-goals, making the training much less reliant on large amounts of preferences to be accurate and more intuitive for the overseer while being computationally more efficient.

Besides, it has several advantages. 1) It is more natural and faster for a human to compare a pair of sub-goals that coming with an optimal sub-goal. 2) When a new sub-goal is added to memory, it only requires to compare the new sub-goal with current best sub-goals. 3) Humans with partial knowledge about the task (i.e. non-experts) can provide feedback signal since they can provide information about pairs of sub-goals. That is, if an expert has poor knowledge about one pair of sub-goals, it can still provide information about other pairs of sub-goals. On the other hand, in standard imitation learning, the human demonstrator has to have some good familiarity with the task in order to select the next optimal sub-goal or action. As a result, we found that a non-expert can provide valuable feedback that can be used to expand the possible applications of deep reinforcement learning agents.

### 4.1.2 Optimizing the meta-controller

We can train the meta-controller $\mu$ based on high-level preference elicitation provided by the predictor network. At time-step $t$, the meta-controller network receives the current state $s_t$ and recommends the next sub-goal $g$. In this paper, we consider that, along with the extrinsic reward, the meta-controller also receives a *preference* bonus $b_t$ that rewards the meta-controller for agreeing with human preferences. Please note that since the reward function $b_t$ is often non-stationary, it is useful to normalize the rewards so that the action-value function can learn quickly [12]. We normalize the preference reward by dividing it by a running estimate of the standard deviations of the preference returns $R_p$, $\overline{\hat{p}(s_t, g|\theta^*)} = [\frac{b_t}{\sigma(R_p)}]$. Thus, the reward $r_t^e$ that receives the meta-controller for selecting the sub-goal $g$ in a state $s_t$ is the following:

$$r_t^e = \beta \cdot R_e + (1-\beta) \cdot [\frac{b_t}{\sigma(R_p)}] = \beta \cdot R_e + (1-\beta) \cdot \overline{\hat{p}(s_t, g|\theta^*)} \quad (3)$$

where $R_e$ is the extrinsic return obtained by selecting the sub-goal $g$ and then following the subpolicy $\pi_g$, $\beta$ is a hyperparameter to weight the importance of the preference bonus, and $\overline{\hat{p}(s_t, g|\theta^*)}$ is the normalized reward calculated by the predictor network. Please note that this formulation accounts for the possibility that human preferences can be suboptimal. Rather than always selecting the preference-based sub-goal, we provide a bonus to the meta-controller. This enables the meta-controller to correct possible human mistakes based on interactions with the environment and discover alternative options that are unknown to the demonstrator.

### 4.1.3 Selecting queries

One common solution to decide how to query preferences relies on ensemble-based uncertainty estimates, as done by Christiano et al. (2017) [16]. However, such an approach tends to be computationally expensive [28]. Moreover, we found this strategy less accurate when operating in the low data regime (with very few preferences), as in our work.

Therefore, we introduce a different approach to decide how to query preferences based on the predictor's confidence. This prompts the predictor to query preferences only for sub-goals that are found with a low degree of confidence. The predictor network's confidence can be modeled using bayesian models. However, in the context of RL, their computational cost can be prohibitive. This problem can be mitigated by using an estimation of Bayesian inference. It was shown that the use of dropout can be interpreted as a Bayesian approximation of Gaussian process [23]. Therefore, we introduce a dropout layer before every weight layer of our predictor network. After each episode, to estimate predictive confidence $c(s, g)$ of a tuple $(s, g)$, we collect the results of stochastic forward passes through the predictor network and then measure the variance in the prediction:

$$c(s, g) = \mathbb{E}_{d_j \sim D}[\hat{p}^{d_j}(s, g|\theta^*) - p]^2 \tag{4}$$

where $\hat{p}^{d_j}(s, g|\theta^*)$ represents the model's prediction with dropout mask $d_j$, $D$ is a set of dropout masks, and $p$ is the predictive posterior mean, $p = \mathbb{E}_{d_j \sim D}\hat{p}^{d_j}(s, g|\theta)$. Since the forward passes can be done concurrently, the method results in a running time identical to that of standard dropout. We can expect the variance of unknown and far-away tuples to be larger than known tuples. Furthermore, using a large number of dropout masks makes this approach much more accurate than training an ensemble of models, while preventing overfitting of recorded preferences.

A score above a threshold $t_{qry}$ results in a query; however, one remaining question is the choice the sub-goal $g_2$ that will be sent to be compared to $g$. $g_2$ is chosen as the second most likely sub-goal to be selected by the meta-controller in the state $s$. The intuition behind this strategy is that rather than comparing $g$ to all the other sub-goals, we can leverage the knowledge of the meta-controller about similar situations to reduce the search space. In other words, even though the meta-controller has a low confidence, it can still discard very unlikely sub-goals and assign higher probabilities to sub-goals similar to the ones encountered before. After a query $(s, g, g_2)$, the resulting feedback is added to the buffer $R$. Then, we finetune the predictor network on $R$.

## 4.2 Combining low-level policies with curiosity

At the low-level, we train subpolicies $\pi_g$ where $g$ is the sub-goal pursued by the subpolicy. In particular we can use any standard off-policy reinforcement learning algorithm like DDPG where each learner accumulates its own experience. Similarly to prior work on hierarchical reinforcement learning [31, 33], we assume access to a *terminal(s,g)* function that indicates the termination of the sub-task and a *done(s,g)* function that indicates a failure or success of the goal being pursued. Hence, we can derive a pseudo-reward function $r^{e,g}$. The subpolicies are trained to maximize this excepted pseudo-reward, defined as follows:

$$r^{e,g} = \begin{cases} 1 & \text{if done(s,g)} \\ -1 & \text{if terminal(s,g)} \wedge \neg\text{done(s,g)} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

However, this pseudo-reward function $r^{e,g}$ may be very sparse depending on the time-horizon of the sub-task. This will become pressing when attempting to scale this method to practical tasks where the number of steps to reach the sub-goal being pursued can be large or the sub-task be very challenging. In this paper, we extend the formulation of the pseudo-reward to tackle sparse reward sub-tasks, drastically reducing the number of interactions to learn subpolicies.

To this end, we propose to make use of a curiosity-based intrinsic reward, $r^i$. This bonus is summed up with the sub-task reward (i.e. pseudo-reward), making rewards dense and more suitable for learning, without the need for handcrafting a reward function for each sub-goal.

Overall, at every time step $t$, $\pi_g$ receives the following reward:

$$r_t = r_t^{e,g} + \alpha \cdot r_t^i \tag{6}$$

where $\alpha$ is a hyperparameter of our method to weight reward components, $r^{e,g}$ is the pseudo-reward, and $r^i$ is the intrinsic reward. In all our experiments, we use Random Network Distillation (RND) [13] as formulation of curiosity, which consists of two neural networks. A fixed network takes an observation to an embedding $f : \mathcal{S} \rightarrow \mathbb{R}^k$ and a reconstructor networks that aims to predict the output of $f$ on the current observation, $\hat{f} : \mathcal{S} \rightarrow \mathbb{R}^k$. Please note that only the reconstructor network is trained by gradient descent to minimize the prediction distance with $f$. The intrinsic reward $r^i$ is calculated via:

$$r_t^i = ||\hat{f}(s_{t+1}) - f(s_{t+1})||^2 \tag{7}$$

An important "trick" that we found is to use a global estimator of curiosity rather than an estimator for each subpolicy. Intuitively, the agent does not need to revisit states that have already been visited by other subpolicies. Furthermore, using a global curiosity estimator ensures that only relevant parts of the state space are collected by the agent. That is, the agent collects experience that has never been explored (or potentially not fully explored) by other subpolicies and that are relevant for reaching the goal being pursued, $g$.

### 4.2.1 Curiosity-driven sub-goal discovery

Rather than relying on manually created sub-goals as done in most prior work, we introduce *Curiosity-Driven Sub-goal Discovery* (CSD) to automatically discover sub-goals with a minimal computational overhead. Intuitively, it has been shown that spikes in the intrinsic curiosity mostly correspond to meaningful events [10, 13]. For instance, in Montezuma's Revenge, large spikes correspond to events such as *passing an obstacle*, *picking an object*, *interacting with a torch*, or *using a ladder*. Therefore, we can hypothesize that some of the states where curiosity spikes can be considered as potential sub-goals. Moreover, we would like to emphasize that irrelevant sub-goals will be discarded (i.e. not selected) by the high-level policy.

The proposed method works as follows. We consider $\Omega$ a set of $M$ prior novelty models (e.g. RND [13]). To fill $\Omega$, every $T$ time steps we substitute the oldest model in memory with the current model. By doing so, we can measure the novelty progress at different time-scales. Novelty-progress $\rho(s_t)$ in a state $s_t$ can now be estimated by measuring the average distance between the current model and previous models at different time-scales:

$$\rho(s_t) = \frac{1}{|\Omega|} \sum_{\theta_{old} \in \Omega} \Big[ ||\hat{f}(s_{t+1}|\theta) - f(s_{t+1}|\theta)||^2 - ||\hat{f}(s_{t+1}|\theta_{old}) - f(s_{t+1}|\theta_{old})||^2 \Big] \quad (8)$$

where $||\hat{f}(s_{t+1}|\cdot) - f(s_{t+1}|\cdot)||^2$ is the novelty estimation parametrized by a set of trainable parameters $\theta$ (current model) or $\theta_{old}$ (prior model). Please note that our method adds states as sub-goals only if they have been visited several times (i.e. regularly visited). This eliminates states that initially have a high novelty but that will be considered less significant by the agent as it explores and gains knowledge about the task. In contrast, task-relevant states are by nature frequently visited. After the novelty progress score computation, a new sub-goal with a score in the lowest 5-percentile is added to memory if the similarity with any other sub-goals is smaller than a similary threshold $b_{similarity}$. This check is necessary for the following reason: the threshold $b_{similarity}$ induces a discretization in the sub-goal space, which enables to store "distinct enough" sub-goals.

## 5 Experiments

In this section, we first describe implementation details and the tasks to be completed by the agent. Then, we answer the following questions:

- Is HhP robust to noisy human preferences?
- Is HhP robust to non-expert preferences?
- What is the impact of using a synthetic oracle?
- Is automatic sub-goal discovery an efficient way to design sub-goals?
- How much do preferences and curiosity help?
- What is the impact of the query budget on the performance?
- What is the impact of the number of experts on the performance?

Finally, we conduct experiments in multiple tasks from the Minigrid environment, MuJoCo suite, and Atari benchmark suite (Fig. 3).

### 5.1 Implementation details

In this section, we refer to our algorithm as hierarchical human preferences (HhP). In all the experiments, the observations are given in the form of images. The RGB images are converted to 84×84 grayscale images. The input given to the policy networks (high-level and low-level) consists of the current observation concatenated with

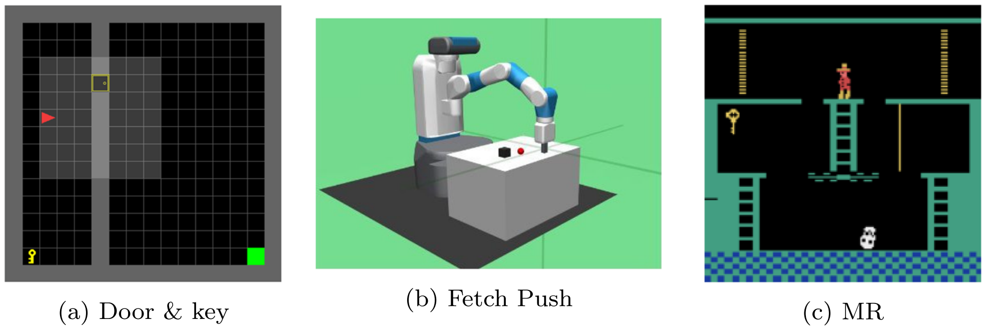

(a) Door & key (b) Fetch Push (c) MR

Fig. 3 Frames from door & key, fetch push, and montezuma's revenge (MR)

the previous three frames. The data buffers contain the most recently encountered tuples. Once their capacity is reached, we substitute the oldest element in memory with the current element. Note that the number of subpolicies is not capped. Empirically, capping the number of subpolicies leads to a deterioration in the overall performance as the agent tends to forget prior skills. When we add a new sub-goal to the set of sub-goals, we randomly initialize the corresponding subpolicy. We use DDQN with prioritized experience replay (Minigrid and Atari; with prioritization exponent $\alpha = 0.6$ and importance sampling exponent $\beta = 0.4$) or DDPG (MuJoCo) at the low level and high level of our algorithm, and RND with similar hyperparameters as in the original implementation [13]. The target network used in Q-learning is updated every 2000 steps.

In order to select the hyperparameters used for the tasks, we ran a grid search with the ranges shown in Section 5.2. We also ran grid searches over the learning rate $\in$ [0.0001, 0.0005, 0.001, 0.005], the number of hidden units $\in$ [128, 256], the query threshold $t_{qry} \in$ [0.20, 0.26, 0.42, 0.60], the number of dropouts $\in$ [500, 1000], and $p \in$ [0.05, 0.1, 0.2]. We did not tune other hyperparameters. As the Mingrid and Mujoco environments are procedurally generated we performed tuning on the validation set, disjoint with the training set. For Atari games, the validation set and training set use different seeds that were randomly chosen (i.e. without any seed tuning). When tuning, we consider the mean final reward of 10 training runs with the same set of hyperparameters as the objective. We employ the following neural network architectures and settings:

- At the high-level, the predictor network uses a siamese architecture [65] with two branches (one for the state and one for the goal), each branch consists in a sequence of three convolutional layers with (32,64,64) filters each, stride: 4,2,1, kernel size of: $8 \times 8, 4 \times 4, 3 \times 3$, and padding 1. We apply a rectifier non-linearity after each convolutional layer. The outputs of the last convolutional layers are concatenated and passed to a serie of two fully connected layers of size 256. The last layer uses a sigmoid activation. After each episode, if the agent made at least one query to the demonstrator, we perform 4 epochs of training on the preference buffer, $R$. Note that the preference buffer has a capacity equal to the query budget.
- The synthetic oracle has the same architecture and hyperparameters as the predictor network. However, it is trained to predict the average return so the last layer is followed by a rectifier non-linearity activation. After each episode, we perform 4 epochs of training on a buffer of size 500K.
- The meta-controller uses a similar architecture, but the dimension of its last layer is the number of subpolicies. It uses a softmax activation to predict the next sub-goal. In our implementations, each goal has a unique *ID* ([0,...,$|G|$) that is assigned after a new goal is selected by the agent. Please note that the input of the predictor network is not the ID of the goal but the goal itself (i.e. an image). Training is carried after each episode and we use $N_{opt} = 4$ for training the meta-controller. The size of the replay buffer $R_H$ is 500K.
- The sub-policies consist in a sequence of three convolutional layers with (32,64,64) filters each, stride: 4,2,1, kernel size of: $8 \times 8, 4 \times 4, 3 \times 3$, and padding 1. We apply a rectifier non-linearity after each convolutional layer. The outputs of the last convolutional layers are concatenated and passed to a serie of one fully connected layer of size 256. The output size is the number of actions. The low-level buffers have a capacity of 50K.

For all networks, training is carried out with a fixed learning rate of $10^{-3}$ using the Adam optimizer [30], with a batch size of 128. The frequency update $T$ of the prior model buffer, $\Omega$, is set to 15,000, and we found $M = 4$ to be sufficient. For Minigrid tasks, we use $t_{qry} = 0.26$. For Atari games and Mujoco tasks, we set $t_{qry} = 0.42$. We set the coefficient of rewards $\beta = 0.7$ and $\alpha = 0.5$. In MuJoCo we set the query budget to 750 and 3000 in Atari games. The query budget is the total number of tuples $(s, g1, g2)$ that are requested to the expert throughout the training process. Once the budget is exhausted, the agent cannot make additional queries. In Minigrid, we report results for different query budgets, between 50 and 200. We set $b_{similarity} = 0.5$. We estimate the sum of discounted preference rewards using a discount factor of 0.99 on rollouts of length 64. $\sigma(R_p)$ (3) is defined as the running average of the standard deviations of these discounted preference rewards. We use the original *terminal* functions provided by the environments. To detect the correct termination/attainment of a subgoal in Atari games, we use a similar strategy as done in hg-Dagger [33]. They count the number of pixels inside of a prespecified box (i.e. around the agent) that has changed in value. In detail, a sub-goal completion is detected if at least 20% of pixels in the landmark's detector box change. For Mingrid tasks, we directly compare the sub-goal and current state. For Fetch tasks, a sub-goal completion is detected if the Euclidean distance to the goal is less or equal to 0.05. In all our experiments, predictor confidence is estimated based on 500 dropout masks with $p = 0.1$. We embrace the single-scale SSIM metric [60] for measuring the similarity between sub-goals, which compares corresponding pixels and their neighborhoods in two images with three metrics: luminance, contrast, and structure. We found this metric to perform better than the standard Euclidean distance when working in environments with complex visual patterns or small details.

**Preferences** In our experiments, feedback is provided by humans (3 annotators) who are being asked to compare pairs of sub-goals. They were recruited among friends and colleagues. They had no special expertise about the tasks, however, before providing feedback to the AI, demonstrators were asked to test the task for 1 hour to get a sense of how it works. Each sub-goal is represented by an image, which allows the demonstrator to quickly compare sub-goals. The human then indicates which sub-goal is favored, that the two sub-goals are equally good, or that the demonstrator is unable to compare the two sub-goals. The human demonstrator had to reply within 5 seconds. Tasks were randomly assigned to the demonstrators and they could not communicate together, which entails that there was no agreement between them. We also run experiments (see Section 5.2.3) using a synthetic oracle whose preferences are generated based on the extrinsic rewards. The synthetic oracle consists of a neural network that estimates the average return (with no discount factor) of a sub-goal. That is, it takes as input a pair of state and sub-goal, and outputs the average return (based on the agent's experience) obtained by an agent starting in this state and aiming to reach this sub-goal. The preferred sub-goal is then the sub-goal with the highest estimated return. In practice (see Section 5.2.3), the reward function provides enough information to the synthetic oracle for selecting accurate preferences.

**Environments** We conduct experiments on several sparse reward environments:

- Minigrid: In Minigrid [15], the world is a partially observable grid. Each tile in the grid contains nothing or one object: ball, box, door, wall, or key. An observation consists of the visible cells surrounding the agent. The agent can choose among seven possible actions: turn left, turn right, move forward, pick up an object, drop the object being carried, open a door, and complete the task. The agent will remain in the same state if the action is not legal. For instance, if there is no key in front of the agent, the agent will remain in the same state when trying to *pick up an object*. The Door & Key task consists of two rooms connected by a door. The agent has to pick up the key in order to unlock the door and then get to the goal. In KeyCorridor, the task is similar to Door & Key but there are multiple rooms and multiple doors. In detail, the agent has to pick up the key, use the key to open the locked door, and pick up the ball. In Multiroom, the agent has to open a serie of doors to reach the final goal. In ObstrMaze, the doors are locked, the keys are hidden in boxes and doors are obstructed by balls. Solving such sparse tasks is challenging since the object locations are randomized and the agent only receives a positive reward $+1$ when it reaches the final goal. These tasks also require sequential decision making (e.g. saving the key to open a distant door) to reach the final goal.
- MuJoCo: In the MuJoCo environment, the agent controls a 7-DoF Sawyer arm which is simulated using the MuJoCo physic engine [57]. A state of the system is represented by an RGB image. The goals are randomly sampled by the environment. They describe the desired position of the object (a box or a puck depending on the task) with some fixed tolerance. The agent receives a positive reward $+1$ after reaching the goal being pursued, 0 otherwise. Action space is 4-dimensional and continuous. Three dimensions specify the desired relative gripper position at the next step. Another dimension specifies the desired distance between the 2 fingers, which are position controlled. The end-effector (EE) is constrained to a 2-dimensional rectangle. We consider four Fetch tasks where the agent controls the robotic arm: (1) Fetch Reach, (2) Fetch Push, (3) Fetch Pick & Place, and (4) Fetch Slide. In Fetch Reach, the agent aims to move the effector to reach a goal as quickly as possible. In Fetch Push, a box is placed on a board in front of the robot and the goal is to move it to the target location on the board. In Fetch Fetch Pick & Place, the agent needs to grab a box and move it to a target location in the air. In Fetch Slide, the robot aims to hit a puck placed on a slippery table so that it reaches a target position located outside the robot's reach.
- Atari: We conduct experiments in the Arcade Learning Environment [8], including: Montezuma's Revenge, Private Eye, Gravitar, Pitfall, Seaquest, and Solaris. These are hard exploration games that might also contain deceptive rewards such as in Pitfall. In these games, the agent has to navigate in complex environments, explore labyrinths, avoid enemies, pass obstacles, and/or collect objects. In Montezuma's Revenge the player explores rooms filled with enemies, obstacles, traps, in an underground labyrinth. In Private Eye, the agent is a private investigator who is driving a car that can move around and jump vertically or over obstacles. The environment can be seen as a labyrinth as it consists of multiple roads located in a city or in the woods. The objective of the game is to capture thieves that sit behind the windows of buildings. In Gravitar, the agent controls a small blue spacecraft. The gravity plays an important role in this game as the ship will be pulled slowly to the bottom of the screen. The agent receives rewards for destroying bunkers. In Pitfall, in

order to recover 32 treasures, the player must maneuver through numerous hazards, including pits, quicksands, and rolling logs. In Seaquest, the agent must shoot enemies to survive. In Solaris, the agent must attempt to keep its fuel consumption low and navigate into hostile battlegroups. Each battlegroup has at least one enemy, which shoots out fuel-sapping drones.

## 5.2 Ablation study

We have conducted ablation studies on some Atari games to investigate: (1) the robustness to imperfect preferences, (2) the robustness to non-expert preferences, (3) the impact of using a synthetic oracle, (4) the impact of the sub-goal discovery strategy, (5) the impact of the components on the performance, (6) the impact of the query budget on the performance, and (7) the impact of the number of experts on the performance.

### 5.2.1 Robustness to imperfect preferences

In the above experiments, we assume perfect preferences (i.e. the demonstrator always provides optimal feedback). However, the teacher might select not the best preferences or even lack knowledge about a sub-goal. We study how our agent performs when imperfect preferences are generated by the human teacher. In order to generate imperfect preferences, we randomly provide a non-optimal preference with a probability $\varrho \in \{0.05, 0.1, 0.2\}$. We report in Table 1 (lines 1-3) the performance of our framework. We observe that HhP can still achieve acceptable performance. For instance, on Montezuma's Revenge, scores obtained by our method remain close to the best scores. Even though the proposed method performs slightly worse in the imperfect setting, it still improves performance as compared to the prior methods. A reason is that the agent can leverage high-level preferences of similar sub-goals and learn from interactions to correct mistakes made by humans. The experimental results demonstrate that our method is reasonably robust to noise in the preferences, and hence a non-expert teacher can provide a feedback signal to the agent.

### 5.2.2 Preferences from a non-expert

In addition to the set of experiments where the agents are trained with imperfect preferences (see Section 5.2.1), we run experiments with a non-expert human. It aims to quantitatively establish how good performs our method when preferences are generated by a non-expert. We only explained the controls of the game to the player and did not elaborate on the specific game mechanics. As shown in Table 2, our method can still perform reasonably well when preferences are generated by a non-expert human. A possible explanation of this result is that the agent can correct possible human mistakes based on its interactions with the environment. In addition, the proposed form of human guidance allows the demonstrator to label tuples as *unable to compare*, which entails that some sub-optimal labels are naturally discarded.

After carrying these experiments, we further asked the non-expert demonstrator to provide optimal actions and preferences for 500 randomly selected tuples $(s, g_1, g_2)$. In other words, the demonstrator had to select the best possible action to take in a state $s$ as well as judge the best sub-goal to select in the same state. Note that for both forms of human guidance, the demonstrator could decide that he is unable to return a label. First, we show how many labels are discarded for both forms of guidance (Table 2 lines 3-4). We then report the frequency of disagreement between the expert and non-expert when feedback is provided (lines 5-6). In all the six games, the non-expert is able to provide more preferences than low-level actions. One possible reason is that less familiarity about the task is necessary to generate high-level knowledge than low-level knowledge. The results also show a much larger frequency of disagreement for

**Table 1** Final mean performance (± std) of our method with various sub-goal creation strategies on Atari games and average success rate (± std) on KeyCorridorS4R3 and Fetch Pick & Place

| | Maximum Mean Score (at convergence) | | | | Success rate | |
|---|---|---|---|---|---|---|
| Method | Montezuma's Revenge | Private Eye | Gravitar | Pitfall | KeyCorridorS4R3 | Pick & Place |
| HhP / $\varrho$=0.05 | 18,657±1,168 | 66,145±1,609 | 2,941±963 | 992±205 | 0.90±0.04 | 0.94 ±0.04 |
| HhP / $\varrho$=0.1 | 16,569±1,453 | 63,693±2,272 | 2,836±1,231 | 759±301 | 0.84±0.07 | 0.79 ±0.07 |
| HhP / $\varrho$=0.2 | 15,028±1,625 | 58,304±2,263 | 2,564±1,456 | 704±416 | 0.77±0.12 | 0.76 ±0.09 |
| HhP / random | 12,698±2,865 | 49,217±6,580 | 2,547±687 | 576±220 | 0.65±0.11 | 0.39 ±0.14 |
| HhP / CSD | 19,914±979 | 68,874±1,265 | 3,100±714 | 1,288±169 | 0.91±0.02 | 0.97 ±0.03 |

Averages over 10 runs are shown after 100M steps (Atari), 5M steps (KeyCorridorS4R3), and 1.5M steps (Fetch Pick & Place)

**Table 2** Final mean performance (mean±std) of our method with expert labels or non-expert labels (lines 1-2)

| | Maximum Mean Score (at convergence) | | | | | |
|---|---|---|---|---|---|---|
| Method | Montezuma's Revenge | Private Eye | Gravitar | Pitfall | Seaquest | Solaris |
| HhP (expert) | **19,914** ± **979** | **68,874** ± **8,265** | **3,100** ± **714** | **1,288** ± **169** | **17,143** ± **1,231** | **3,996** ± **511** |
| HhP (non-expert) | 16,125 ± 687 | 57,090 ± 9,124 | 3,081 ± 801 | 1,022 ± 364 | 16,062 ± 1,478 | 3,747 ± 419 |
| # Actions | 118 | 132 | 87 | 147 | 61 | 85 |
| # Preferences | 29 | 45 | 12 | 21 | 32 | 30 |
| Disagreement (actions) | 0.12 | 0.21 | 0.13 | 0.15 | 0.19 | 0.09 |
| Disagreement (Preferences) | 0.05 | 0.07 | 0.04 | 0.08 | 0.11 | 0.03 |

We report the results of our method achieved over total 100M timesteps of training, averaged over 10 seeds. We also report the number of discarded labels (lines 3-4) and the frequency of disagreement between the expert and the non-expert (lines 5-6)

actions than preferences. We found that it is generally more intuitive for a human to judge outcomes than controlling the agent. We conclude that our method tends to be more intuitive and easier to demonstrate for a non-expert than standard demonstrations.

### 5.2.3 Synthetic oracle for preference elicitation

The majority of the experiments use a human for labeling. We also run experiments with a synthetic oracle. This experiment aims to quantitatively establish how good is our method when preferences are generated by the agent itself. With synthetic labels, poor performance can stem from two causes: (1) failure of the oracle to capture a good estimation of value function, resulting in bad preferences, or (2) conservative exploration behaviors. Table 3 summarizes the overall performance of each setup. In most tasks, performance is similar, but in Montezuma's Revenge and Pitfall, human preferences are clearly better. This is due to the very long time-horizon of these tasks, which makes the estimation of the value function difficult. Overall, it shows that our method can be generally trained using synthetic labels.

### 5.2.4 Impact of the sub-goal discovery strategy

To see the potential benefits of using an automatic sub-goal creation strategy, we explore the performance of HhP with sub-goals created using the following strategies: randomly discovered, and automatically discovered (CSD) (Table 1, line 4-5). Note that in complex and temporally-extended tasks like Atari games, manually creating all sub-goals is deemed infeasible. During late training, the agents trained with CSD always reached higher performance. It validates that our strategy can discover a diverse range of sub-goals that cover key events of the task. On the other hand, randomly selecting sub-goals deteriorates the performance. Nevertheless, even with randomly discovered sub-goals, the meta-controller and the teacher can still discard irrelevant sub-goals and select meaningful sub-goals, keeping the performance in an acceptable range. However, the final performance of HhP trained with randomly discovered sub-goals is capped since the agent cannot keep learning once all the meaningful sub-goals are mastered.

To further validate the depicted strategy, we show the sub-goals discovered in the first room of Montezuma's Revenge and compare it against the optimal sub-goals

**Table 3** Final mean performance (mean±std) of our method with synthetic labels or human labels

| | Maximum Mean Score (at convergence) | | | | | |
|---|---|---|---|---|---|---|
| Method | Montezuma's Revenge | Private Eye | Gravitar | Pitfall | Seaquest | Solaris |
| HhP (synthetic) | 13,125 ± 1,612 | 46,601 ±6,459 | 2,947 ±830 | 611 ± 154 | 14,286 ± 1,029 | 3,804 ± 524 |
| HhP (human) | **19,914** ± **979** | **68,874** ± **8,265** | **3,100** ± **714** | **1,288** ± **169** | **17,143** ± **1,231** | **3,996** ± **511** |

We report the results of our method achieved over total 100M timesteps of training, averaged over 10 seeds

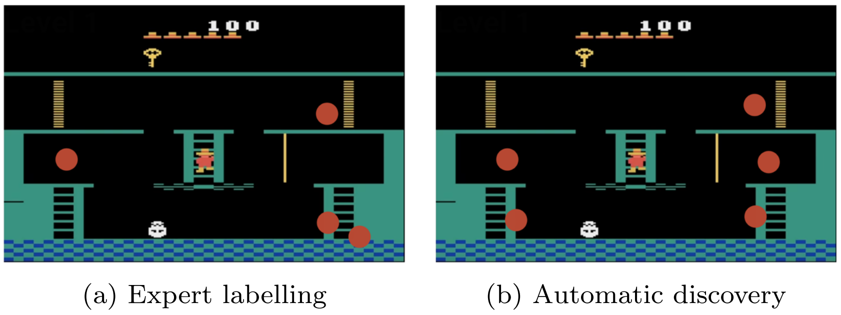


**Fig. 4** Screenshots from Montezuma's Revenge: manually created sub-goals (left), and automatically discovered sub-goals (right). The sub-goals (the agent's position) are denoted by a red dot

created by an expert [33]. For simplicity, we report in Fig. 4 the location of the agent for each sub-goal. We can see that the automatically discovered sub-goals are very similar with manually created sub-goals. For instance the sub-goal *pick a key* was found by both strategies. A noticeable difference is that our strategy found a larger number of sub-goals compared to experts. Thus, this experiment validates that CSD can be used as a robust alternative to manually created sub-goals.

### 5.2.5 Impact of the components on the performance

In order to measure how much do high-level preferences and curiosity help, we compare the following experimental setups:

- Curiosity (Cur): The model is trained solely based on its own curiosity.
- Preferences (Pref): The model is trained from human high-level preferences, without curiosity.
- Full model (HhP): The model is trained from human high-level preferences and curiosity.

As can be observed in Fig. 5, *Cur* produces faster learning during the early stages of the agent's training. For instance, on Pitfall after 50M training steps the agent equipped with curiosity achieves a score of $\approx 260$. However, as selecting the next sub-goal becomes increasingly difficult, learning without human preferences hurts the performance. We generally observe that curiosity significantly reduces training time of subpolicies, but has a limited impact on the gain in performance. On the other hand, learning from human high-level preferences (*Pref*) shows large improvements compared to *Cur* - human preferences help the agent to rapidly order the subpolicies. Nevertheless, learning without curiosity increases the exploration workload, requiring

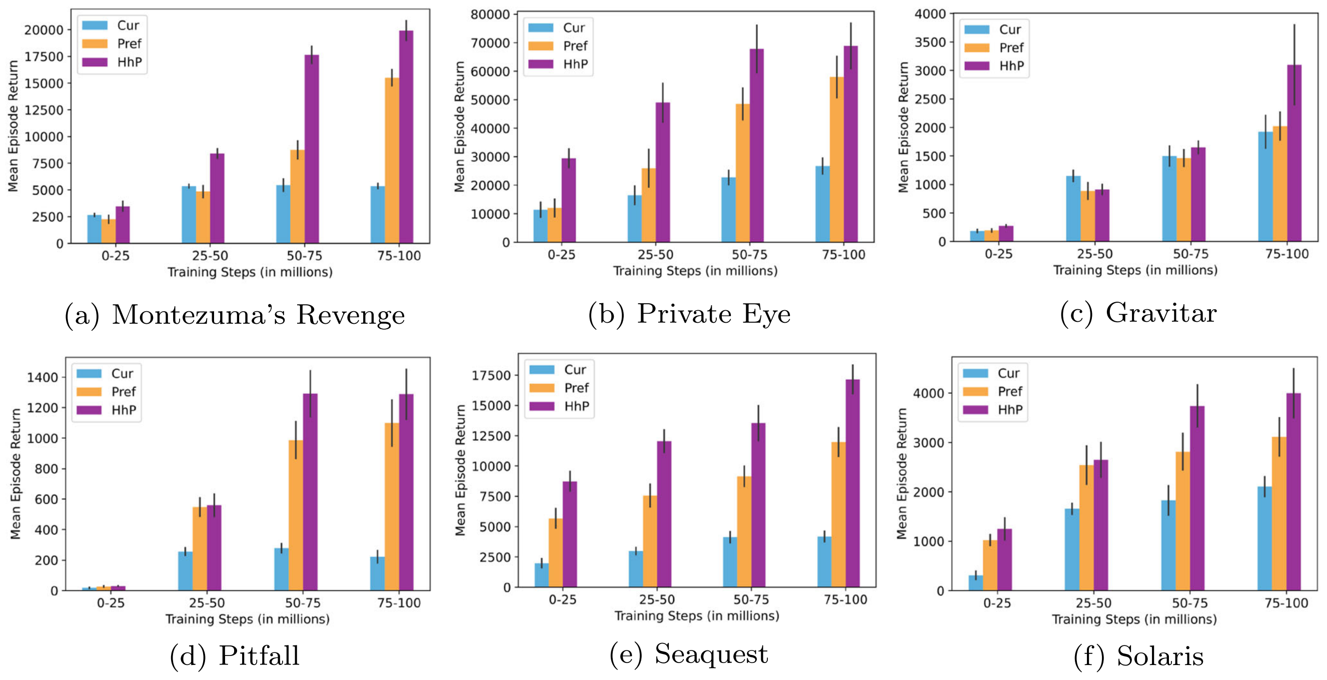


**Fig. 5** Performance for different experimental setups on 6 Atari games after 100M agent steps. Performance of the model (HhP) against the model trained solely based on its own curiosity (Cur) and with preferences (Pref). Results are averaged over 10 runs (±std)

more interactions to reach similar performance as HhP. This experiment demonstrates that curiosity-driven exploration plays a central role in reducing the number of trial-and-errors and human guidance enables various forms of common sense reasoning, improving the overall performance.

### 5.2.6 Query budget of preferences

We now report evaluations showing the effect of increased query budget. Figure 6 demonstrates that agents trained with a larger query budget obtain higher mean returns after similar numbers of updates. However, despite a small query budget (less than 4000), our method can still learn near-optimal policies. We can draw the observation that as the query budget increases, the learning effect on the agent gradually improves. Nonetheless, for the results with 3000 and 6000 queries, we can see that even though the number of queries significantly differs, the difference in learning effect can be negligible. This can happen when the queried preferences cover a broad enough number of sub-goals and therefore the agent does not need to make additional queries. As a result, our method leverages a small amount of preferences that cover critical sub-goals, leading to dramatic reductions in both human effort and cost of exploration.

### 5.2.7 Number of experts

Finally, one legitimate question is to study the impact of the number of different demonstrators on the agent's performance. Ideally, the policy performance should not be too sensitive to this hyperparameter. Especially, using multiple demonstrators should not decrease the performance of our agent. We perform a study for a various number of demonstrators $1, 2, 3$ on Montezuma's Revenge and Private Eye. Note that using *1 expert* means that all the preferences are queried to the same human. On the other hand, when more than one expert is involved, each run is randomly assigned to an expert. Table 4 shows that the HhP performance is robust to the choice of this hyperparameter. In the future, we anticipate using some forms of inter agreement between the experts to improve the quality of the preferences. In our experiments, preferences are queried to one expert without considering his knowledge about the situation, however, we can expect improvements by selecting the expert that is the most familiar with the current situation.

**Table 4** Final mean performance (mean±std) of our method for a various number of experts

| Maximum Mean Score (at convergence) | | |
|---|---|---|
| Method | Montezuma's Revenge | Private Eye |
| HhP (1 expert) | 20,005 ± 964 | 66,741 ± 7,459 |
| HhP (2 experts) | 19,321 ± 805 | 65,256 ± 8,198 |
| HhP (3 experts) | 19,914 ± 979 | 68,874 ± 8,265 |

We report the results of our method achieved over total 100M timesteps of training, averaged over 10 seeds

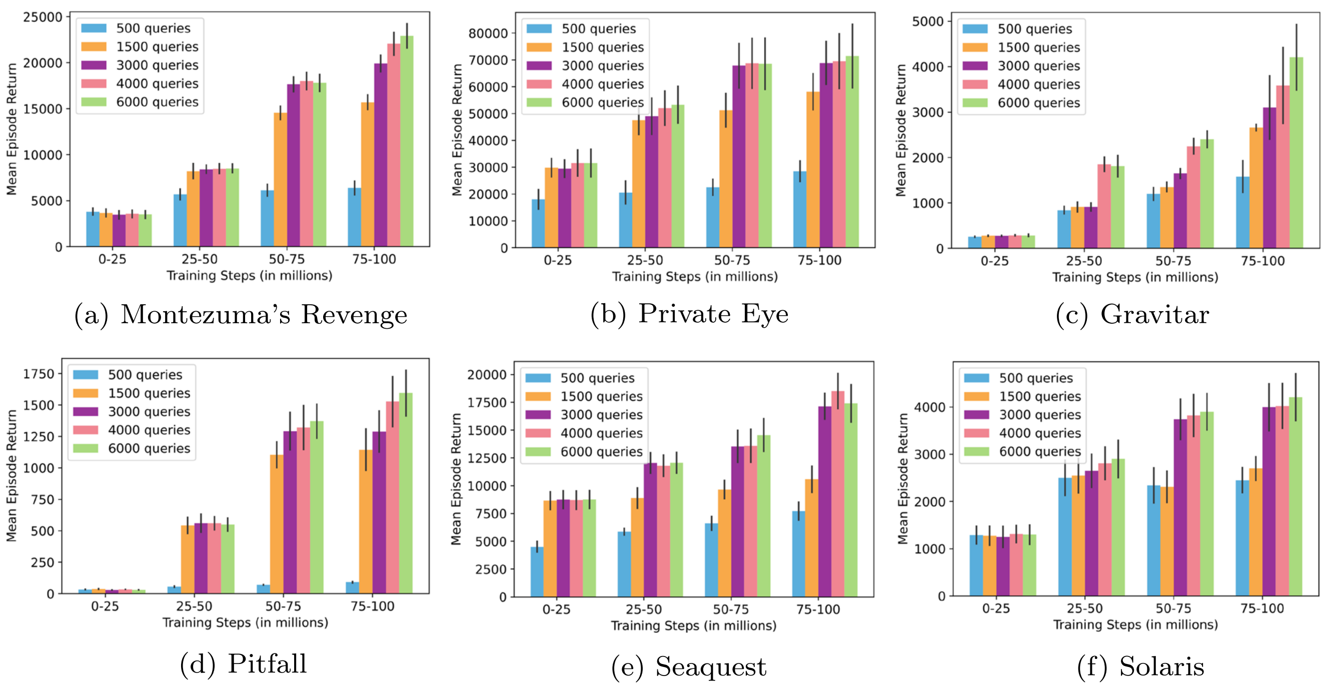


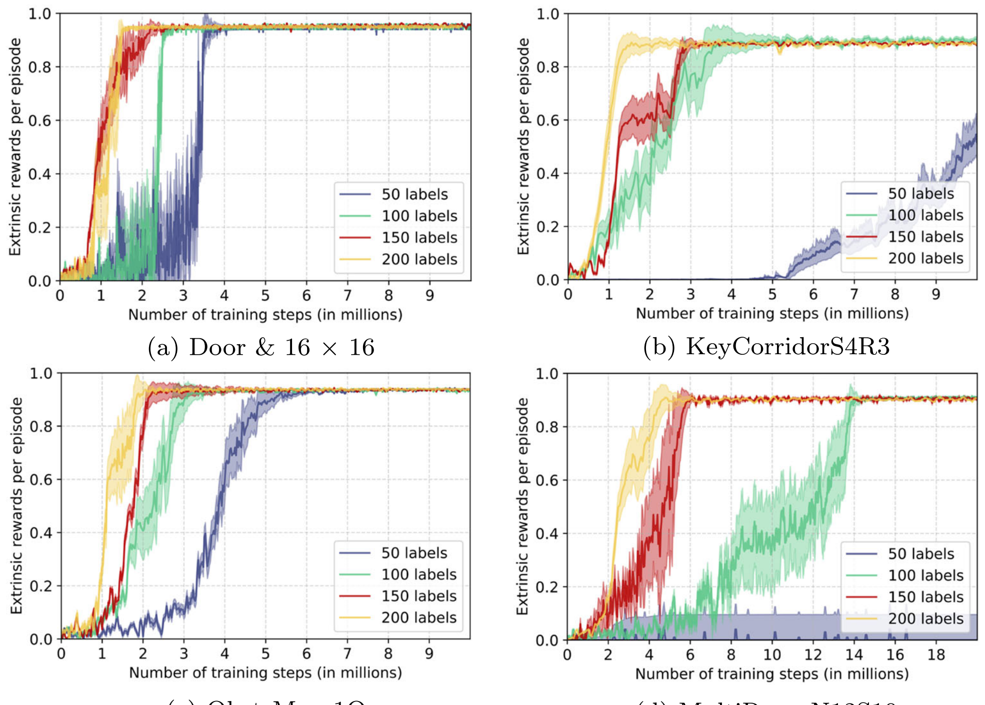
**Fig. 6** Performance of our method on 6 Atari games after 100M agent steps, for different query budgets. Results are averaged over 10 runs (±std)

## 5.3 Procedurally generated tasks

We now perform experiments on a set of four procedurally generated tasks in the Minigrid environment to evaluate the overall performance of our algorithm and its generalization ability to unseen views or appearances. Please note that due to a large number of random environment instances, this domain requires a very large number of samples for tabular algorithms. Figure 7 depicts the training performance on Door & 16 × 16, KeyCorridorS4R3, ObstrMaze1Q, and MultiRoomN12S10. The results of each run are averaged to provide a mean curve in each figure, and the standard error is used to make the shaded region surrounding each curve. In all cases, the use of the proposed method results in significant improvements in the performance of the policy, leading to near-optimal policies. Figure 7 shows that all the tasks can be solved with very few queries (i.e. less than 200 queries). As can be further seen, on Door & 16 × 16 and ObstrMaze1Q, even though HhP (50 queries) primarily progresses more slowly than HhP trained with a larger query budget (HhP (150 queries), HhP (200 queries)), it is ultimately capable of achieving similar performance. Overall, leveraging the proposed form of human guidance - high-level preferences, enables our agent to rapidly reach good performance and generalize the provided domain knowledge to unseen appearances.

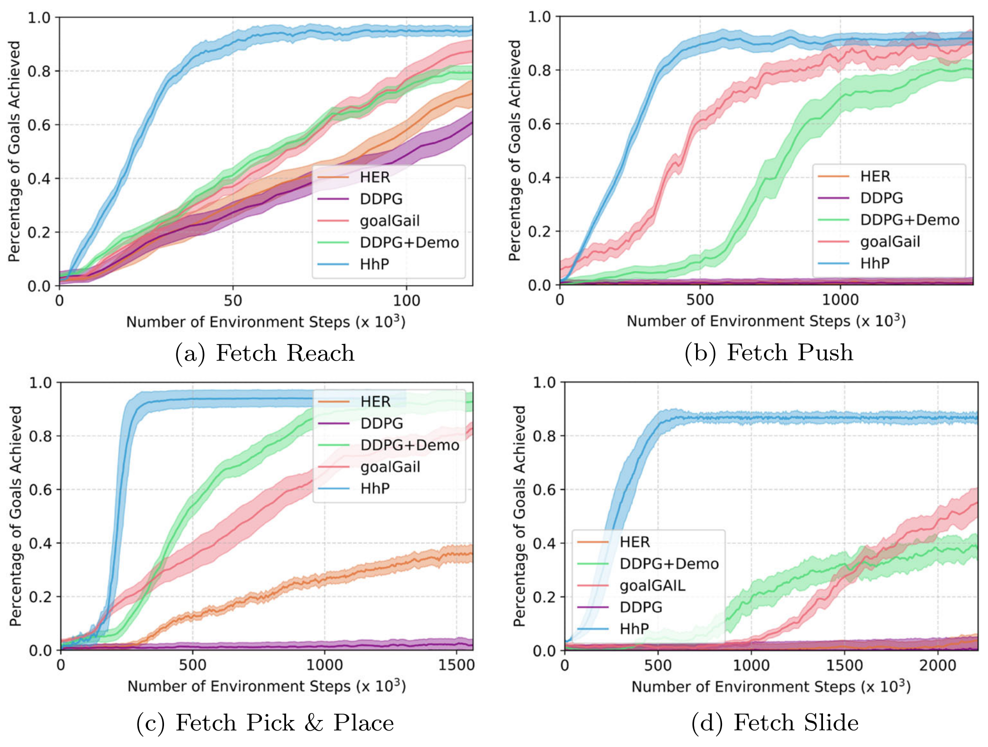
## 5.4 Robotic tasks

In this section we evaluate the agent on four different tasks (Fetch) from the robotic domain built on top of MuJoCo: Fetch Reach, Fetch Push, Fetch Pick and Place, and Fetch Slide. We compare our method against several baselines including DDPG [35], HER [3], DDPG+Demo [39], and goalGail [19]. We show learning curves in Fig. 8. Our method can learn comparable or superior policies using a significantly smaller number of human feedback than goalGAIL and DDPG+Demo. For instance, on Pick and Place, on average only 156 queries were made by HhP. As excepted, it ends up reaching similar final performance, however, our method has a faster convergence rate, reducing the amount of required interactions with the environment. Furthermore, unlike our algorithm, DDPG+Demo passively access the demonstration data, so we actively provide help in the form of high-level preferences to our agent when it struggles. That is, the agent receives human feedback when its confidence in the sub-goal selection is low. At the low-level, we make use of curiosity to guide the agent's learning. We can further observe that incrementally querying preferences over sub-goals keeps the number of required preferences very low, while enabling non-expert demonstrations - it is more intuitive for a human to judge in which direction to move the robotic arm than controlling the robotic arm (see Section 5.2.2). We conclude that our



(a) Door & 16 × 16 (b) KeyCorridorS4R3 (c) ObstrMaze1Q (d) MultiRoomN12S10

**Fig. 7** Performance of HhP for different numbers of queries (labels) on a variety of procedurally generated tasks in MiniGrid: Door & 16 × 16, KeyCorridorS4R3, ObstrMaze1Q, and MultiRoomN12S10. All curves (mean±std) are averaged over 10 random runs



(a) Fetch Reach (b) Fetch Push (c) Fetch Pick & Place (d) Fetch Slide

**Fig. 8** Learning curves averaged over 10 runs for different models: HhP, DDPG, HER, DDPG+Demo, and goalGail. The models are trained on robotic tasks from the Fetch environment

method provides the capability to effectively learn from multiple types of internal and external supervision.

## 5.5 Hard exploration games

We also test the proposed method on six difficult exploration Atari 2600 games from the Arcade Learning Environment (ALE) [8]: Montezuma's Revenge, Private Eye, Gravitar, Pitfall, Seaquest, and Solaris. In the selected games, training an agent with a poor exploration strategy often results in a suboptimal policy. We compare our method to the performance of PPO [47], PPO+RND [13], DQfD [25], Imitation [25], Pref (No-Demo) [16], and Pref (Demo) [28]. The results are shown in Table 5. We consider the final mean performance of 10 training runs with the same set of hyperparameters. It is observed the plain PPO algorithm obtained a score close to zero and could not solve most of the tasks. On the other hand, a hierarchical decomposition of the tasks drastically reduces the number of required interactions. We also found that human preferences is vital - even with significantly more labels, DQfD fails to reach scores comparable to our method. We hypothesize that

**Table 5** Final mean performance (mean±std) of our method and baselines on Atari games

| | Maximum Mean Score (at convergence) | | | | | |
|---|---|---|---|---|---|---|
| Method | Montezuma's Revenge | Private Eye | Gravitar | Pitfall | Seaquest | Solaris |
| PPO [47] | 1,259±610 | 50±37 | 1,826±255 | -21±6 | 664±258 | 1,021 ±199 |
| PPO+RND [13] | 8,152±653 | 8,666±1051 | **3,906±246** | -3±1 | 3,179±378 | 3,282±281 |
| DQfD [25] | 4,739 | 40,908 | 1,693 | 57 | 12,361 | 2,616 |
| Imitation [25] | 576 | 43,047 | 248 | 182 | 195 | 3,589 |
| Pref (No-Demo) [16] | 23 ± 5 | 256 ± 69 | – | – | 1,011 ± 216 | – |
| Pref (Demo) [28] | 2,829 ± 714 | 50,159 ± 11,657 | – | – | 515 | 142 |
| Average Human [61] | 4,753 | 69,571 | 3,351 | 6,464 | 20,182 | 12,327 |
| HhP (ours) | **19,914 ± 979** | **68,874 ± 8,265** | 3,100 ± 714 | **1,288 ± 169** | **17,143 ± 1,231** | **3,996 ± 511** |

We report the results of our method achieved over total 100M timesteps of training, averaged over 10 seeds. Some historical papers did not consider games, in which case the score is displayed as "-"

human demonstrations often deviate from the agent's goal and do not enable efficient common sense reasoning - they cannot be used by the agent for planning or reasoning but are limited to specific situations. Please note that Gravitar is an exception, the curiosity-based approach PPO+RND is hard to beat. On Private Eye, our method achieves a mean score higher than Pref (Demo) trained without curiosity. Overall, on the six tasks, using human high-level guidance enables our agent to achieve various forms of common sense reasoning and avoids learning from scratch, surpassing standard reinforcement learning baselines. Moreover, even with access to a smaller number of labels, our method outperforms other preference-based methods that make use of low-level preferences. One reason is that HhP requires much less human queries to efficiently drive the agent's learning. We observed a reduction by a factor of $\approx 2$ over models that use low-level preferences (Pref(No-Demo), Pref(Demo)). The depicted method also drastically reduces the total amount of feedback compared to DQfD on Montezuma's Revenge (17949), Private Eye (10899), Gravitar (15377), Pitfall (35347), Seaquest (57453), and Solaris (28552). We should also emphasize that comparing pairs of sub-goals appears to be easier than providing an optimal action or comparing low-level trajectories (see Section 5.2.2). Moreover, comparing pairs of sub-goals is more *efficient per second of human time* compared to standard preference-based approaches that require the demonstrator to visualize trajectory segments. For instance, in Pref (No-Demo) the demonstrators responded to the average query in 3-5 seconds [16]. On the other hand, on average, the demonstrator responded to our queries in 1.6 seconds. Besides, we further observed that using curiosity is critical for reducing the number of low-level interactions, especially in tasks with complex dynamics like Montezuma's Revenge or Private Eye.

## 6 Future research direction

In this paper, we presented *hierarchical learning from human preferences and curiosity*, a method introducing human preferences at the high-level along with curiosity at the low-level. Precisely, to greatly reduce the human involvement, we proposed to introduce human guidance in the form of high-level preferences between sub-goals, enabling non-expert feedback and alleviating the need for demonstrating complex behaviors. Further benefits stem from efficiently learning subpolicies by leveraging an intrinsic reward. Unlike traditional hierarchical learning methods where the agent passively receives a set of sub-goals, our method actively discovers sub-goals based on the agent's curiosity progress - *curiosity-driven sub-goal discovery*, which eliminates the need to handcraft sub-goals. In the depicted work, human preferences provide prior assumptions about the domain to the agent, avoiding learning from scratch and facilitating various forms of common sense reasoning. Precisely, grafting human preferences onto the agent allows it to plan a sequence of sub-goals based on its own common sense priors (without having to experience a situation). On the other hand, curiosity is used for control of sub-tasks too challenging for even humans to perform well. We demonstrated the effectiveness of our approach and compared it against several baselines on Minigrid, MuJoCo, and Atari. Our method shows substantial improvements over prior work in terms of average scores and exploration efficiency. Moreover, even very small amounts of preferences let us outperform prior imitation-based approaches on multiple sparse reward tasks.

That being said, we acknowledge that our approach has certain limitations. The proposed method relies on a *terminal* and a *done* function. Such functions are widely used in prior hierarchical learning work [31, 33]. However, it would be interesting to let the agent decide when to change of active sub-goal. Another solution would be to learn a function that decides when the active sub-goal is terminated or done.

A key element in our method is leveraging human preferences for selecting sub-goals. In the long run, it would be desirable to introduce a predicate that requests the teacher a new optimal next sub-goal. In other words, if there is no optimal next sub-goal (among $\mathcal{G}$), it would be important to request a different sub-goal designed by a teacher. Although automatically discovering sub-goals that cover a wide range of options is relatively easy in most tasks, we cannot expect our method to always discover all meaningful options.

Another research direction is how to create preference in order to further reduce human effort. In our experiments, we compared human labels to a synthetic oracle. While human labels consistently produced higher scores, our method could still outperform most prior work when trained from synthetic labels. One promising direction is to replace the human expert by another agent's advice. After the learner determines when to query a prospective teacher, it may be possible to query another agent that has already acquired knowledge about the situation. This inter-agent teaching strategy has been used to solve tasks such as video game playing [56], but it remains an open problem in complex tasks [17]. Another possible solution could be to train the agent on a mixture of preference data collected from an agent and a human. We believe that exploring multi-agent collaboration is an important direction in order to further reduce human involvement, making the proposed approach easier to apply to real-world domains.

Finally, when running experiments, we needed to choose a threshold to ensure storing significantly different sub-goals. Even though we achieved improvements on most tasks with a fixed value, it requires little tuning of hyperparameters. One way to overcome the need for parameter tuning is to incorporate the idea of discriminability like done in DIAYN [20], but we leave it to future work to explore this direction further. Another similar avenue for future research is to incorporate an adaptive confidence threshold to further improve query selection.

## References

1. Abbeel P, Ng AY (2004) Apprenticeship learning via inverse reinforcement learning. In: Proceedings of the international conference on machine learning. p 1
2. Andreas J, Klein D, Levine S (2017) Modular multitask reinforcement learning with policy sketches. In: International conference on machine learning. pp 166–175
3. Andrychowicz M, Wolski F, Ray A, Schneider J, Fong R, Welinder P, McGrew B, Tobin J, Abbeel OP, Zaremba W (2017) Hindsight experience replay. In: Advances in neural information processing systems. pp 5048–5058
4. Argall BD, Chernova S, Veloso M, Browning B (2009) A survey of robot learning from demonstration. Robot Auton Syst 57(5): 469–483
5. Bacon PL, Harb J, Precup D (2017) The option-critic architecture. In: Proceedings of the AAAI conference on artificial intelligence, vol 31
6. Baranes A, Oudeyer PY (2013) Active learning of inverse models with intrinsically motivated goal exploration in robots. Robot Auton Syst 61(1):49–73
7. Bellemare M, Srinivasan S, Ostrovski G, Schaul T, Saxton D, Munos R (2016) Unifying count-based exploration and intrinsic motivation. In: Proceedings of advances in neural information processing systems. pp 1471–1479
8. Bellemare MG, Naddaf Y, Veness J, Bowling M (2013) The arcade learning environment: An evaluation platform for general agents. J Artif Intell Res 47:253–279
9. Bougie N, Ichise R (2020a) Exploration via progress-driven intrinsic rewards. In: Proceedings of the international conference on artificial neural networks, vol 22, pp 269–281
10. Bougie N, Ichise R (2020b) Fast and slow curiosity for high-level exploration in reinforcement learning. Appl Intell
11. Bougie N, Cheng LK, Ichise R (2018) Combining deep reinforcement learning with prior knowledge and reasoning. ACM SIGAPP Appl Comput Rev 18(2):33–45
12. Burda Y, Edwards H, Pathak D, Storkey A, Darrell T (2019a) Large-scale study of curiosity-driven learning. In: Proceedings of the the international conference on learning representations
13. Burda Y, Edwards H, Storkey A, Klimov O (2019b) Exploration by random network distillation. In: Proceedings of the international conference on learning representations
14. Chernova S, Veloso M (2007) Confidence-based policy learning from demonstration using gaussian mixture models. In: Proceedings of the international joint conference on autonomous agents and multiagent systems. pp 1–8
15. Chevalier-Boisvert M, Willems L, Pal S (2018) Minimalistic gridworld environment for openai gym. https://github.com/maximecb/gym-minigrid
16. Christiano PF, Leike J, Brown T, Martic M, Legg S, Amodei D (2017) Deep reinforcement learning from human preferences. In: Advances in neural information processing systems. pp 4299–4307
17. Da Silva FL, Warnell G, Costa AHR, Stone P (2020) Agents teaching agents: a survey on inter-agent transfer learning. Auton Agent Multi-Agent Syst 34(1):1–17
18. Dietterich TG (2000) Hierarchical reinforcement learning with the maxq value function decomposition. J Artif Intell Res 13:227–303
19. Ding Y, Florensa C, Abbeel P, Phielipp M (2019) Goal-conditioned imitation learning. In: Advances in neural information processing systems. pp 15298–15309
20. Eysenbach B, Gupta A, Ibarz J, Levine S (2019) Diversity is all you need: Learning skills without a reward function. In: International conference on learning representations
21. Florensa C, Held D, Geng X, Abbeel P (2018) Automatic goal generation for reinforcement learning agents. In: International conference on machine learning. pp 1515–1528
22. Fruit R, Lazaric A (2017) Exploration-exploitation in mdps with options. In: Artificial intelligence and statistics. pp 576–584
23. Gal Y, Ghahramani Z (2016) Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In: Proceedings of the international conference on machine learning. pp 1050–1059
24. Garnelo M, Arulkumaran K, Shanahan M (2016) Towards deep symbolic reinforcement learning. arXiv:http://arxiv.org/abs/160905518
25. Hester T, Vecerik M, Pietquin O, Lanctot M, Schaul T, Piot B, Horgan D, Quan J, Sendonaris A, Osband I, Dulac-Arnold G, Agapiou J, Leibo JZ, Gruslys A (2018) Deep q-learning from demonstrations. In: Annual meeting of the association for the advancement of artificial intelligence
26. Ho J, Ermon S (2016) Generative adversarial imitation learning. In: Advances in neural information processing systems. pp 4565–4573
27. Hsu D (2019) A new framework for query efficient active imitation learning. arXiv:http://arxiv.org/abs/191213037
28. Ibarz B, Leike J, Pohlen T, Irving G, Legg S, Amodei D (2018) Reward learning from human preferences and demonstrations in atari. In: Advances in neural information processing systems. pp 8011–8023
29. Kendall MG, Smith BB (1940) On the method of paired comparisons. Biometrika 31(3/4):324–345
30. Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. arXiv:http://arxiv.org/abs/14126980
31. Kulkarni TD, Narasimhan K, Saeedi A, Tenenbaum J (2016) Hierarchical deep reinforcement learning: Integrating temporal

abstraction and intrinsic motivation. In: Advances in neural information processing systems. pp 3675–3683
32. Laversanne-Finot A, Péré A, Oudeyer PY (2021) Intrinsically motivated exploration of learned goal spaces. Front Neurorobot 14:109
33. Le H, Jiang N, Agarwal A, Dudik M, Yue Y, Daumé HIII (2018) Hierarchical imitation and reinforcement learning. In: Proceedings of machine learning research, pp 2917–2926
34. Levy A, Konidaris G, Platt R, Saenko K (2017) Learning multi-level hierarchies with hindsight. arXiv:http://arxiv.org/abs/171200948
35. Lillicrap TP, Hunt JJ, Pritzel A, Heess N, Erez T, Tassa Y, Silver D, Wierstra (2015) Continuous control with deep reinforcement learning. arXiv:http://arxiv.org/abs/150902971
36. Machado MC, Bellemare MG, Bowling M (2018) Count-based exploration with the successor representation. arXiv:http://arxiv.org/abs/180711622
37. Mathewson KW, Pilarski PM (2017) Actor-critic reinforcement learning with simultaneous human control and feedback. arXiv:http://arxiv.org/abs/170301274
38. Nachum O, Gu S, Lee H, Levine S (2018) Data-efficient hierarchical reinforcement learning. arXiv:http://arxiv.org/abs/180508296
39. Nair A, McGrew B, Andrychowicz M, Zaremba W, Abbeel P (2018) Overcoming exploration in reinforcement learning with demonstrations. In: Proceedings of the IEEE international conference on robotics and automation. pp 6292–6299
40. Ng AY, Russell SJ, et al. (2000) Algorithms for inverse reinforcement learning. In: Proceedings of the international conference on machine learning. pp 663–670
41. Ostrovski G, Bellemare MG, van denOordA, Munos R (2017) Count-based exploration with neural density models. In: Proceedings of the international conference on machine learning. pp 2721–2730
42. Pathak D, Agrawal P, Efros AA, Darrell T (2017) Curiosity-driven exploration by self-supervised prediction. In: International conference on international conference on machine learning. pp 2778–2787
43. Pomerleau DA (1991) Efficient training of artificial neural networks for autonomous navigation. Neural Comput 3(1):88–97
44. Röder F, Eppe M, Nguyen PD, Wermter S (2020) Curious hierarchical actor-critic reinforcement learning. arXiv:http://arxiv.org/abs/200503420
45. Saunders W, Sastry G, Stuhlmueller A, Evans O (2018) Trial without error: Towards safe reinforcement learning via human intervention. In: Proceedings of the international conference on autonomous agents and multiagent systems. pp 2067–2069
46. Savinov N, Raichuk A, Marinier R, Vincent D, Pollefeys M, Lillicrap T, Gelly S (2019) Episodic curiosity through reachability. In: Proceedings of the international conference on learning representations
47. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O (2017) Proximal policy optimization algorithms. arXiv:http://arxiv.org/abs/170706347
48. Shon AP, Verma D, Rao RP (2007) Active imitation learning. In: Proceedings of the AAAI conference on artificial intelligence. pp 756–762
49. Silver D, Huang A, Maddison CJ, Guez A, Sifre L, Van Den Driessche G, Schrittwieser J, Antonoglou I, Panneershelvam V, Lanctot M, et al. (2016) Mastering the game of go with deep neural networks and tree search. Nature 529(7587):484
50. Stone P, Veloso M (2000) Layered learning. In: European conference on machine learning. Springer, pp 369–381
51. Strehl AL, Littman ML (2008) An analysis of model-basedinterval estimation for markov decision processes. J Comput Syst Sci 74(8):1309–1331
52. Sutton RS (1988) Learning to predict by the methods of temporal differences. Machine Learn 3(1):9–44
53. Sutton RS, Barto AG (1998) Reinforcement learning: an introduction. MIT press, Cambridge
54. Sutton RS, Precup D, Singh S (1999) Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. Artif Intell 112(1-2):181–211
55. Tang H, Houthooft R, Foote D, Stooke A, Chen X, Duan Y, Schulman J, De TurckF, Abbeel P (2017) # exploration: a study of count-based exploration for deep reinforcement learning. In: Proceedings of the 31st international conference on neural information processing systems. pp 2750–2759
56. Taylor ME, Carboni N, Fachantidis A, Vlahavas I, Torrey L (2014) Reinforcement learning agents providing advice in complex video games. Connect Sci 26(1):45–63
57. Todorov E, Erez T, Tassa Y (2012) Mujoco: A physics engine for model-based control. In: 2012 IEEE/RSJ international conference on intelligent robots and systems. pp 5026–5033
58. Vecerik M, Hester T, Scholz J, Wang F, Pietquin O, Piot B, Heess N, Rothörl T, Lampe T, Riedmiller M (2017) Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. arXiv:http://arxiv.org/abs/170708817
59. Vezhnevets AS, Osindero S, Schaul T, Heess N, Jaderberg M, Silver D, Kavukcuoglu K (2017) Feudal networks for hierarchical reinforcement learning. In: Internationasearning. pp 3540–3549
60. Wang Z, Bovik AC, Sheikh HR, Simoncelli EP (2004) Image quality assessment: from error visibility to structural similarity. IEEE Trans Image Process 13(4):600–612
61. Wang Z, Schaul T, Hessel M, Van HasseltH, Lanctot M, De Freitas N (2016) Dueling network architectures for deep reinforcement learning. In: International conference on machine learning. pp 1995–2003
62. Warnell G, Waytowich N, Lawhern V, Stone P (2018) Deep tamer: Interactive agent shaping in high-dimensional state spaces. In: Thirty-Second AAAI conference on artificial intelligence. pp 1545–1554
63. Wilson A, Fern A, Tadepalli P (2012) A bayesian approach for policy learning from trajectory preference queries. In: Advances in neural information processing systems. pp 1133–1141
64. Wirth C, Akrour R, Neumann G, Fürnkranz J (2017) A survey of preference-based reinforcement learning methods. J Mach Learn Res 18(1):4945–4990
65. Zagoruyko S, Komodakis N (2015) Learning to compare image patches via convolutional neural networks. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp 4353–4361
66. Zhang R, Torabi F, Guan L, Ballard DH, Stone P (2019) Leveraging human guidance for deep reinforcement learning tasks. In: Proceedings of the international joint conference on artificial intelligence. pp 6339–6346
67. Zhang X, Ma H (florensa2018automatic) Pretraining deep actor-critic reinforcement learning algorithms with expert demonstrations. arXiv:http://arxiv.org/abs/180110459
68. Ziebart BD, Maas A, Bagnell JA, Dey AK (2008) Maximum entropy inverse reinforcement learning. In: Proceedings of the national conference on artificial intelligence. pp 1433–1438
69. Zuo G, Zhao Q, Lu J, Li J (2020) Efficient hindsight reinforcement learning using demonstrations for robotic tasks with sparse rewards. Int J Adv Robot Syst 17

**Nicolas Bougie** graduated from the University of Paris Sud, France in 2017. He studied about machine learning and artificial intelligence. He is currently a PhD student at the National Institute of Informatics in Japan and a student at the Sokendai University. His research area covers reinforcement learning, deep learning and machine learning.

**Ryutaro Ichise** received his Ph.D. degree in computer science from Tokyo Institute of Technology, Tokyo, Japan, in 2000. From 2001 to 2002, he was a visiting scholar at Stanford University. He is currently an associate professor in Principles of Informatics Research Division at National Institute of Informatics in Japan. His research interests include machine learning, semantic web, and data mining.