A. Proofs

Proof of Theorem 2. The first term TC^I_{FULL} should be obvious as the expert inspects the agent's overall behavior in each episode. Whenever something goes wrong in an episode, the expert labels the whole trajectory, incurring C^L_{FULL} each time. The remaining work is to bound the number of episodes where agent makes one or more mistakes. This quantity is bounded by the number of total mistakes made by the halving algorithm, which is at most the logarithm of the number of candidate functions (policies), $\log|\Pi_{\text{FULL}}| = \log\left(|\mathcal{M}||\Pi_{\text{LO}}|^{|\mathcal{G}|}\right) = \log|\mathcal{M}| + |\mathcal{G}|\log|\Pi_{\text{LO}}|$. This completes the proof. □

Proof of Theorem 1. Similar to the proof of Theorem 2, the first term TC^I_{FULL} is obvious. The second term corresponds to the situation where $\texttt{Inspect}_{\text{FULL}}$ finds issues. According to Algorithm 2, the expert then labels the subgoals and also inspects whether each subgoal is accomplished successfully, which incurs $C^L_{\text{HI}} + H_{\text{HI}}C^I_{\text{LO}}$ cost each time. The number of times that this situation happens is bounded by (a) the number of times that a wrong subgoal is chosen, plus (b) the number of times that all subgoals are good but at least one of the subpolicies fails to accomplish the subgoal. Situation (a) occurs at most $\log|\mathcal{M}|$ times. In situation (b), the subgoals chosen in the episode must come from $\mathcal{G}_{\text{opt}}$, and for each of these subgoals the halving algorithm makes at most $\log|\Pi_{\text{LO}}|$ mistakes. The last term corresponds to cost of $\texttt{Label}_{\text{LO}}$ operations. This only occurs when the meta-controller chooses a correct subgoal but the corresponding subpolicy fails. Similar to previous analysis, this situation occurs at most $\log|\Pi_{\text{LO}}|$ for each "good" subgoal ($g \in \mathcal{G}_{\text{opt}}$). This completes the proof. □

B. Additional Experimental Details

In our experiments, *success rate* and *external rewards* are reported as the trailing average over previous 100 episodes of training. For hierarchical imitation learning experiments in maze navigation domain, the success rate is only measured on separate test environments not used for training.

In addition to experimental results, in this section we describe our mechanism for subgoal detection / terminal predicate for Montezuma's Revenge and how the Maze Navigation environments are created. Network architectures from our experiments are in Tables 1 and 2.

B.1. Maze Navigation Domain

We compare hg-DAgger/Q with the hierarchical reinforcement learning baseline (h-DQN, Kulkarni et al., 2016) with the same network architecture for the meta-controller and subpolicies as hg-DAgger/Q and similarly enhanced Q-learning procedure.

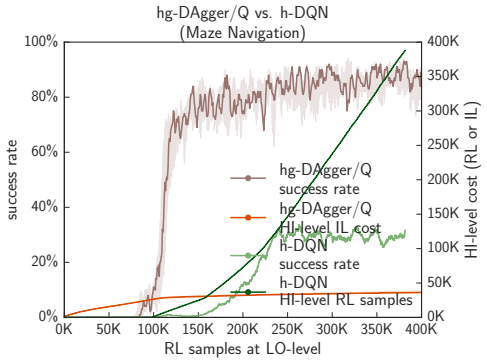

Figure 4. Maze navigation: hybrid IL-RL (full task) versus h-DQN (with 50% head-start).

Similar to the Montezuma's Revenge domain, h-DQN does not work well for the maze domain. At the HI level, the planning horizon of 10–12 with 4–5 possible subgoals in each step is prohibitively difficult for the HI-level reinforcement learner and we were not able to achieve non-zero rewards within in any of our experiments. To make the comparison, we attempted to provide additional advantage to the h-DQN algorithm by giving it some head-start, so we ran h-DQN with 50% reduction in the horizon, by giving the hierarchical learner the optimal execution of the first half of the trajectory. The resulting success rate is in Figure 4. Note that the hybrid IL-RL does not get the 50% advantage, but it still quickly outperforms h-DQN, which flattens out at 30% success rate.

B.1.1. Creating Maze Navigation Environments

We create 2000 maze navigation environments, 1000 of which are used for training and 1000 maps are used for testing. The comparison results for maze navigation (e.g., Figure 2) are all based on randomly selected environments among 1000 test maps. See Figure 5 for additional examples of the environments created. For each map (environment instance), we start with a 17×17 grid, which are divided into 4×4 room structure. Initially, no door exists in between rooms. To create an instance of the maze navigation environment, the goal block (yellow) and the starting position are randomly selected (accepted as long as they are not the same). Next, we randomly select a wall separating two different room and replace a random red block (lava) along this wall with a door (black cell). This process continues until two conditions are satisfied:

- There is a feasible path between the starting location and the goal block (yellow)
- The minimum distance between start to goal is at least 40 steps. The optimal path can be constructed using

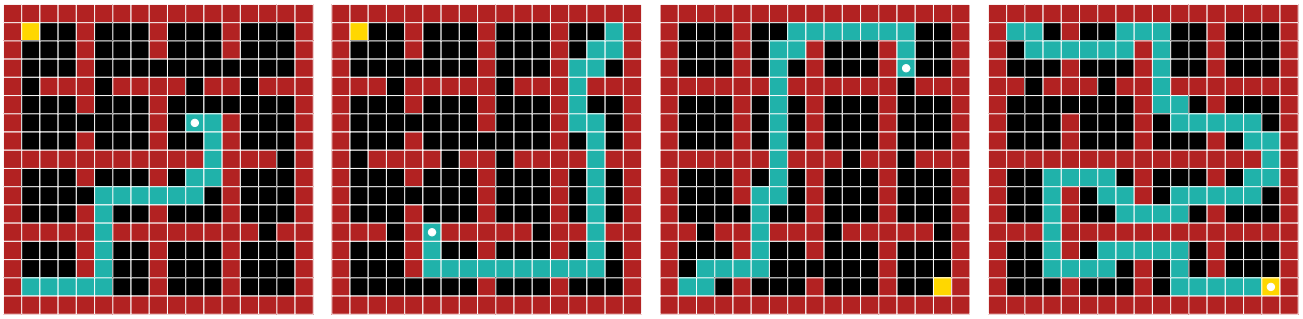

Figure 5. Maze navigation. Sample random instances of the maze domain (different from main text). The 17×17 pixel representation of the maze is used as input for neural network policies.

graph search

Each of the 2000 environments create must satisfy both conditions. The expert labels for each environment come from optimal policy computed via value iteration (which is fast based on tabular representation of the given grid world).

B.1.2. Hyperparameters for Maze Navigation

The network architecture used for maze navigation is described in Table 1. The only difference between subgoal policy networks and metacontroller network is the number of output class (4 actions versus 5 subgoals). For our hierarchical imitation learning algorithms, we also maintain a small network along each subgoal policy for subgoal termination classification (one can also view the subgoal termination classifier as an extra head of the subgoal policy network).

The contextual input (state) to the policy networks consists of 3-channel pixel representation of the maze environment. We assign different (fixed) values to goal block, agent location, agent's trail and lava blocks. In our hierarchical imitation learning implementations, the base policy learner (DAgger and behavior cloning) update the policies every 100 steps using stochastic optimization. We use Adam optimizer and learning rate of 0.0005.

Table 1. Network Architecture—Maze Domain

1: Convolutional Layer	32 filters, kernel size 3, stride 1
2: Convolutional Layer	32 filters, kernel size 3, stride 1
3: Max Pooling Layer	pool size 2
4: Convolutional Layer	64 filters, kernel size 3, stride 1
5: Convolutional Layer	64 filters, kernel size 3, stride 1
6: Max Pooling Layer	pool size 2
7: Fully Connected Layer	256 nodes, relu activation
8: Output Layer	softmax activation (dimension 4 for subpolicy, dimension 5 for meta-controller)

B.2. Montezuma's Revenge

Although the imitation learning component tends to be stable and consistent, the samples required by the reinforcement learners can vary between experiments with identical hyperparameters. In this section, we report additional results of our hybrid algorithm for the Montezuma's Revenge domain.

For the implementation of our hybrid algorithm on the game Montezuma's Revenge, we decided to limit the computation to 4 million frames for the LO-level reinforcement learners (in aggregate across all 4 subpolicies). Out of 100 experiments, 81 out of 100 successfully learn the first 3 subpolicies, 89 out of 100 successfully learn the first 2 subpolicies. The last subgoal (going from the bottom of the stairs to open the door) proved to be the most difficult and almost half of our experiments did not manage to finish learning the fourth subpolicy within the 4 million frame limit (see Figure 7 middle pane). The reason mainly has to do with the longer horizon of subgoal 4 compared to other three subgoals. Of course, this is a function of the design of subgoals and one can always try to shorten the horizon by introducing intermediate subgoals.

However, it is worth pointing out that even as we limit the h-DQN baseline to only 2 subgoals (up to getting the key), the h-DQN baseline generally tends to underperform our proposed hybrid algorithm by a large margin. Even with the given advantage we confer to our implementation of h-DQN, all of the h-DQN experiments failed to successfully master the second subgoal (getting the key). It is instructive to also examine the sample complexity associated with getting the key (the first positive external reward, see Figure 7 right pane). Here the horizon is sufficiently short to appreciate the difference between having expert feedback at the HI level versus relying only on reinforcement learning to train the meta-controller.

The stark difference in learning performance (see Figure 7 right) comes from the fact that the HI-level expert advice effectively prevents the LO-level reinforcement learners from

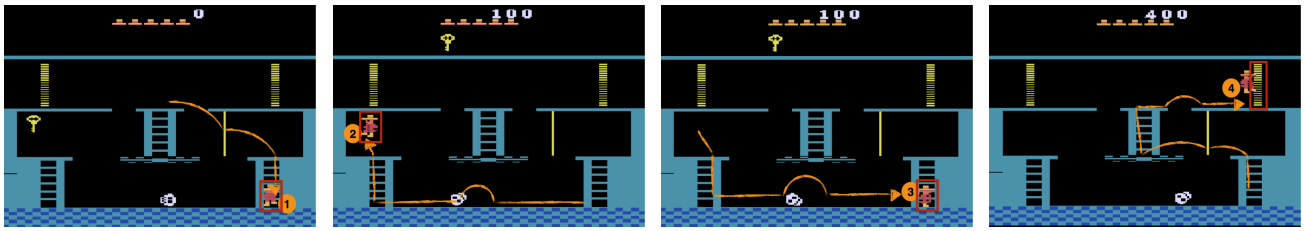

Figure 6. Montezuma's Revenge: Screenshots of the environment with 4 designated subgoals in sequence.

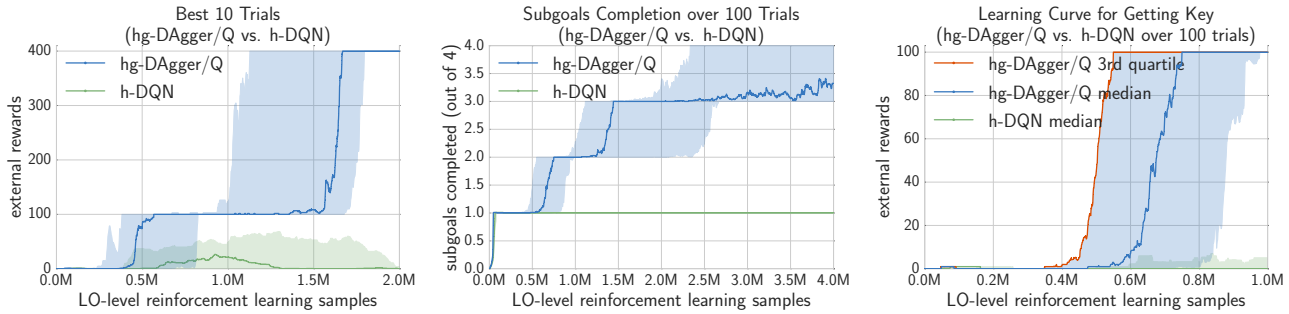

Figure 7. Montezuma's revenge: hybrid IL-RL versus hierarchical RL. (Left) Median reward, min and max across the best 10 trials. The agent completes the first room in less than 2 million samples. The shaded region corresponds to min and max of the best 10 trials. *(Middle)* Median, first and third quartile of subgoal completion rate across 100 trials. The shaded region corresponds to first and third quartile. *(Right)* Median, first and third quartile of reward across 100 trials. The shaded region corresponds to first and third quartile. h-DQN only considers the first two subgoals to simplify the learning task.

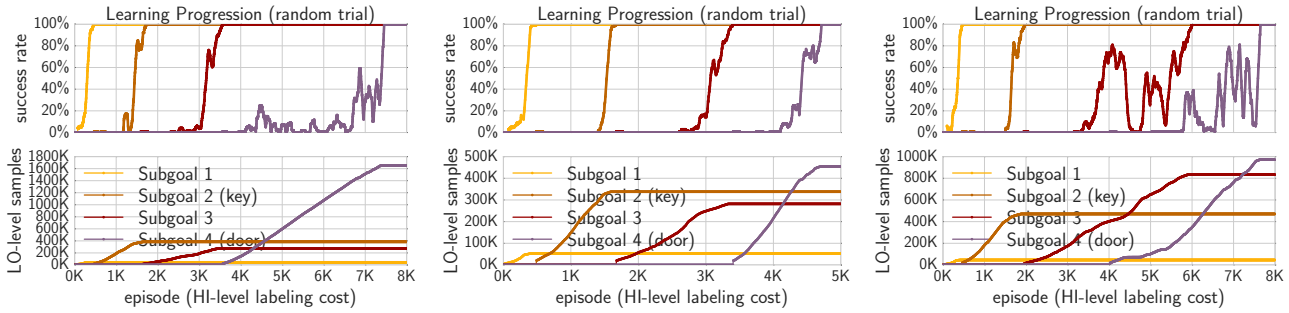

Figure 8. Montezuma's revenge: Learning progression of Algorithm 3 in solving the entire first room. The figures show three randomly selected successful trials.

accumulating bad experience, which is frequently the case for h-DQN. The potential corruption of experience replay buffer also implies at in our considered setting, learning with hierarchical DQN is no easier compared to flat DQN learning. Hierarchical DQN is thus susceptible to collapsing into the flat learning version.

B.2.1. Subgoal Detectors for Montezuma's Revenge

In principle, the system designer would select the hierarchical decomposition that is most convenient for giving feedback. For Montezuma's Revenge, we set up four subgoals and automatic detectors that make expert feedback trivial. The subgoals are landmarks that are described by small rectangles. For example, the door subgoal (subgoal 4) would be represented by a patch of pixel around the right door (see Figure 6 right). We can detect the correct termination / attainment of this subgoal by simply counting the number of pixels inside of the pre-specified box that has changed in value. Specifically in our case, subgoal completion is detected if at least 30% of pixels in the landmark's detector box changes.

B.2.2. Hyperparameters for Montezuma's Revenge

Neural network architecture used is similar to (Kulkarni et al., 2016). One difference is that we train a separate neural network for each subgoal policy, instead of main-

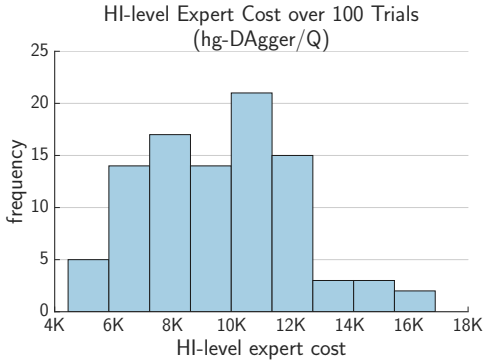

Figure 9. Montezuma's Revenge: Number of HI-level expert labels. Distribution of Hi-level expert labels needed across 100 trials; the histogram excludes 6 outliers whose number of labels exceeds 20K for ease of visualization

Table 2. Network Architecture—Montezuma's Revenge

1: Conv. Layer	32 filters, kernel size 8, stride 4, relu
2: Conv. Layer	64 filters, kernel size 4, stride 2, relu
3: Conv. Layer	64 filters, kernel size 3, stride 1, relu
4: Fully Connected Layer	512 nodes, relu, normal initialization with std 0.01
5: Output Layer	linear (dimension 8 for subpolicy, dimension 4 for meta-controller)

taining a subgoal encoding as part of the input into a policy neural network that shares representation for multiple subgoals jointly. Empirically, sharing representation across multiple subgoals causes the policy performance to degrade we move from one learned subgoal to the next (a phenomenon of catastrophic forgetting in deep learning literature). Maintaining each separate neural network for each subgoal ensures the performance to be stable across subgoal sequence. The metacontroller policy network also has similar architecture. The only difference is the number of output (4 output classes for metacontroller, versus 8 classes (actions) for each LO-level policy).

For training the LO-level policy with Q-learning, we use DDQN (Van Hasselt et al., 2016) with prioritized experience replay (Schaul et al., 2015b) (with prioritization exponent $\alpha = 0.6$, importance sampling exponent $\beta_0 = 0.4$). Similar to previous deep reinforcement learning work applied on Atari games, the contextual input (state) consists of four consecutive frames, each converted to grey scale and reduced to size 84$\times$ 84 pixels. Frame skip parameter as part of the Arcade Learning Environment is set to the default value of 4. The repeated action probability is set to 0, thus the Atari environment is largely deterministic. The experience memory has capacity of 500K. The target network used in Q-learning is updated every 2000 steps. For stochastic optimization, we use rmsProp with learning rate of 0.0001, with mini-batch size of 128.

C. Additional Related Work

Imitation Learning. Another dichotomy in imitation learning, as well as in reinforcement learning, is that of value-function learning versus policy learning. The former setting (Abbeel & Ng, 2004; Ziebart et al., 2008) assumes that the optimal (demonstrated) behavior is induced by maximizing an unknown value function. The goal then is to learn that value function, which imposes a certain structure onto the policy class. The latter setting (Daumé et al., 2009; Ross et al., 2011; Ho & Ermon, 2016) makes no such structural assumptions and aims to directly fit a policy whose decisions well imitate the demonstrations. This latter setting is typically more general but often suffers from higher sample complexity. Our approach is agnostic to this dichotomy and can accommodate both styles of learning. Some instantiations of our framework allow for deriving theoretical guarantees, which rely on the policy learning setting. Sample complexity comparison between imitation learning and reinforcement learning has not been studied much in the literature, perhaps with the exception of the recent analysis of AggreVaTeD (Sun et al., 2017).

Hierarchical Reinforcement Learning. Feudal RL is another hierarchical framework that is similar to how we decompose the task hierarchically (Dayan & Hinton, 1993; Dietterich, 2000; Vezhnevets et al., 2017). In particular, a feudal system has a manager (similar to our HI-level learner) and multiple submanagers (similar to our LO-level learners), and submanagers are given pseudo-rewards which define the subgoals. Prior work in feudal RL use reinforcement learning for both levels; this can require a large amount of data when one of the levels has a long planning horizon, which we demonstrate in our experiments. In contrast, we propose a more general framework where imitation learners can be used to substitute reinforcement learners to substantially speed up learning, whenever the right level of expert feedback is available. Hierarchical policy classes have been additional studied by He et al. (2010), Hausknecht & Stone (2016), Zheng et al. (2016), and Andreas et al. (2017).

Learning with Weaker Feedback. Our work is motivated by efficient learning under weak expert feedback. When we only receive demonstration data at the high level, and must utilize reinforcement learning at the low level, then our setting can be viewed as an instance of learning under weak demonstration feedback. The primary other way to elicit weaker demonstration feedback is with preference-based or gradient-based learning, studied by Fürnkranz et al. (2012), Loftin et al. (2016), and Christiano et al. (2017).