A Appendix

A.1 Related Work

Recent progress in RL has partly been made possible by combining rich function classes like deep neural networks with powerful techniques such as Q-learning [24] and actor-critic approaches [16]. In recent years huge strides have been made in training agents for complex tasks, such as for playing Atari games with discrete and continuous action spaces [12, 25]. Additionally, RL has achieved human-level performance in games like chess [26] and Go [27].

However, it is becoming increasingly clear that simple exploration is not enough to circumvent the curse of dimensionality in environments with long horizons and sparse rewards. As a remedy the broad area of Hierarchical Reinforcement Learning (HRL) attempts to decompose RL problems into multiple levels of abstraction— temporal, spatial, or otherwise. Many works deploy separate policies over different time horizons and action spaces [11, 28, 29, 30]. Temporal abstraction in planning can be traced back at least to Sutton et al. [2], where the options were introduced to refer to lower level policies. In most existing research in hierarchical RL, learning a sub-task precedes the learning of a policy which uses it. This includes [31, 32, 33, 34, 35]. This paper is set apart by the fact it does not rely on an apriori knowledge of the starting distributions of the sub-tasks as required by some related literature on hierarchical RL, such as [36]. We further elaborate on literature relevant to the two important facets of our approach below.

On expert help via primitives or skills: Various recent works utilize expert help in the form of primitives or skills [4, 5, 6, 7, 8, 9, 20]. This takes place via parameterized action spaces [4], stitching together independent task schemas (or skills) [5, 6, 9, 20] or learning parameters of action primitves [7, 9, 8]. In all of these cases, learning takes place within the traditional hierarchical framework, *i.e.*, bottom-up (see Figure 2), where a sub-task is learnt before the policy that uses it. Our method proposes a framework to learn top-down instead with large-improvements in sample-efficiency over traditional bottom-up learning. With minimal changes in the implementation of their primitives and skills, the above diverse strategies can also benefit with shorter learning times by utilizing our framework to learn top-down.

On expert help with humans-in-the-loop: Other approaches to expert intervention include explicit help with humans-in-the-loop required throughout training [21, 22]. In contrast, in our method, human effort is only required at the beginning to create the option template hierarchies.

On hierarchical learning in robotics: Task and motion planning (TAMP) methods [37] are alternate hierarchical approaches for tackling long-horizon problems in robotics. They generally highlight the interplay of motion-level and task-level planning, with the task-level planner constraining the motion-level planner. In contrast, our approach allows the task-level planner to learn using feasible paths which can be satisfied by the motion-level planner. This allows us to achieve high reward with few samples.

On exploration in high-dimensional tasks: An alternative approach to counter the need for exploration in high dimensional tasks is through the use of demonstrations [1] for long horizon planning. The intuition is that introducing a degree of *behavioral cloning* of expert demonstrations helps reduce the amount of exploration the agent has to perform. A different approach known as *Hindsight Experience Replay* (HER) [23] incorporates the goal information into the state, using a failed terminal state as an alternative goal to reward the transitions leading to it. These methods are orthogonal to our approach. By themselves, they are unable to achieve the large degree of reduction in sample-complexity seen with our framework. Yet, alongside expert help, they further improve sample-efficiency as seen in our experiments.

Task	Policy Sketch
get wood	-
get iron	-
make bridge	get wood $\rightarrow$ get iron $\rightarrow$ use factory
get gold	make bridge $\rightarrow$ use bridge on water

Task	Learning level	Option templates
get gold	1	{give bridge, primitive actions}
make bridge	2	{give wood, give iron, primitive actions}
get wood	3	{primitive actions}
get iron	3	{primitive actions}

Table 5: Get gem hierarchical task: **[LEFT]** policy sketches [14] & **[RIGHT]** option templates for each task in the hierarchy. The order of the rows represent the learning order in the two alternatives.

Task	Episodes	
	Curriculum learning [14]	Option templates (Ours)
get gold	$> 2.65 \times 10^6$	10516.0 ± 2833.0
make bridge	$> 1.8 \times 10^6$	7974.0 ± 1853.0

Table 6: Comparison of total episodes (and standard deviations over ten random seeds) to train an agent to solve the *get gem* task via option templates and curriculum learning [14].

A.2 Additional Details of Experiments on the Craft Environment

A.2.1 Results on the *get gold* Task.

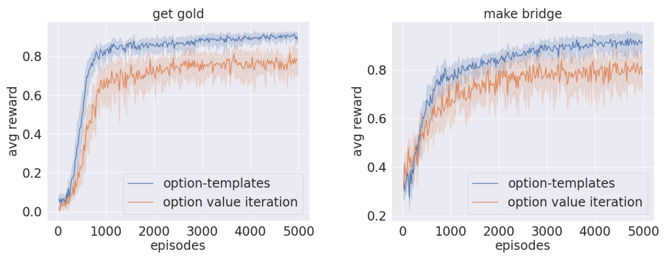

Figure 6: Average reward vs episodes for solving each hierarchical sub-task. We compare option templates (our method) and option-value iteration (baseline) for the task *get gold* in craft environment.

A.2.2 Additional Environment Details

For a fair comparison with [14], we set the maximum steps in the environment for *get wood* and *get iron* to 100; for *make stick* to 200; for *make bridge* to 300; for *make axe* and *get gold* to 400; and for *get gem* to 500.

A.2.3 Hyperparameters

The Adam optimizer was used to train all networks. The batch size was always set to the size of the memory. The exploration (ϵ) is exponentially reduced. For all tasks, we use a discount factor (γ) of 0.99 for primitive actions and 0.99^d for option templates. The exponent (d) is the expected task horizon and is reported in Table 7. Other hyperparameters are also given in Table 7.

In option value iteration, we use the exact task horizon which is available to the agent since options are learnt bottom-up. The other hyperparameters in option value iteration are the same as those reported for option templates.

A.2.4 Discussion

Figures 4 and 6 show that option templates obtain a higher average reward than option-value iteration at all levels. In particular, at level 2, the two curves appear closer since the option templates

Parameters	*get gem*	*make axe*	*make stick*	*get gold*	*make bridge*
expected task horizon (d)	100	50	40	100	50
memory size (transitions)	$20\times d$				
learning frequency (steps)	$20\times d$				
learning rate	0.001				

Table 7: Hyperparameters for option templates in craft environment.

curve converges to a lower value than in levels 1 and 3, likely due to the tasks on level 2 being more difficult. We believe the gap occurs because policies at the levels below the one currently being learned typically do not perform perfectly in option-value iteration. Further, these errors may "confuse" training the option in option-value iteration. Whereas in our approach which uses option templates which do not make these errors, we avoid the aforementioned issue.

A.3 Additional Details of Experiments on the Fetch and Stack Environment

The Fetch and Stack environment is implemented inside the MuJoCo physics engine designed to simulate multi-body interactions including contacts, joints and collision. Unlike craft, this environment does not permit the implementation of option-templates as part of its action space. We circumvent this problem by adding action primitives using simple state-feedback controllers. Each option-template is implemented in this fashion, for different learning levels (see Table 3), until the lowest level is reached. The agent treats the state feedback controllers as option-templates during training phase. As intended, the agent only gets access to read the states after the option-templates reach the termination condition. The option templates can also terminate after a timeout (80 steps for level 1 and 30 steps for level 2).

A.3.1 Hyperparameters

The Adam optimizer was used to train all networks. The batch size was always set to 20% of the memory size. The exploration (ϵ) is exponentially reduced. All other hyperparameters are given in Table 8. The baseline used the same hyperparameters as option templates.

Parameters	level 1	level 2
discount factor (γ)	0.2	0.8
memory size (transitions)	100	
learning frequency	every 1500 steps for 3 blocks every 2000 steps for 4 blocks	
learning rate	0.001	

Table 8: Hyperparameters for option templates in fetch and stack environment.

A.3.2 Comparison to Learning multi-level hierarchies with hindsight [38]

We also compared our method with Levy et al. [38]'s Hindsight Actor Critic (HAC) algorithm in the Fetch and Stack environment on the three block stacking task and clearly notice that it cannot learn to stack even two blocks in more than 10 times our learning time.

HAC [38] obtain an average reward of only 0.138 after $(67.2 \pm 0.1) \times 10^5$ steps (obtained over 5 random seeds). In comparison, we obtain an average reward of 1 after a learning duration of only $(4.5 \pm 0.1) \times 10^5$ timesteps.

We note that, we use the high-level / low-level learning strategy described in Yang et al. [39] and only mention the high-level learning time (which is much lower than the low-level learning time) for Levy et al. [38] here for a fair comparison. We plot the variation of rewards at all levels with our method, baseline and HAC [38] in Figure 7.

Further, the method proposed in Yang et al. [39] called Universal Option Framework is shown to outperform Levy et al. [38]'s Hindsight Actor Critic. Even with this boost, according to Yang et al. [39], their method attains a 0.7 average reward at the high-level only after $(480\pm3.2)\times10^5$ timesteps.

Method	Easy		Medium		Hard	
	avg. goal difference	steps	avg. goal difference	steps	avg. goal difference	steps
Option Templates	5.79 ± 0.92	0.3 M (win game) + 1.8 M (attack) = 2.1 M	3.0 ± 0.71	0.3 M (win game) + 1.8 M (attack) = 2.1 M	2.0 ± 0.18	0.3 M (win game) + 1.8 M (attack) = 2.1 M
Baseline	1.2 ± 0.22	4.2 M	0.8 ± 0.33	4.2 M	0.26 ± 0.1	4.2 M
1-Player Impala [19]	5.14 ± 2.88	500 M	-0.36 ± 0.11	500 M	-0.47 ± 0.48	500 M
1-Player DQN DQN [19]	8.16 ± 1.05	500 M	2.01 ± 0.27	500 M	0.27 ± 0.56	500 M

Table 9: Numerical values of Figure 5c: comparison of time steps and corresponding average goal difference (including standard deviation across 5 random seeds) for option templates (controlling 11-players), baseline (controlling 11-players), and single-player agents [19].

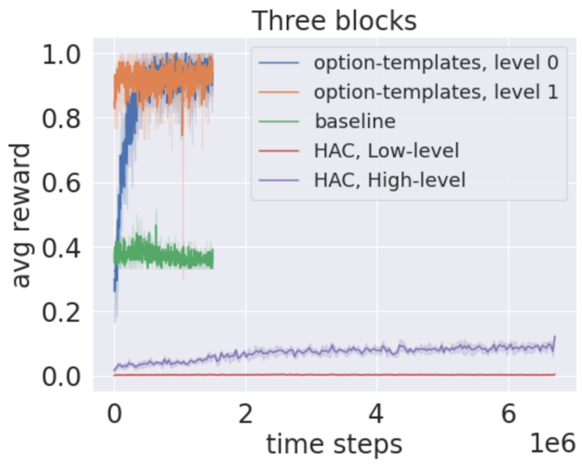

Figure 7: Average reward vs episodes for all levels with option templates (our method), the baseline and HAC [38] in the Fetch and Stack environment.

A.4 Additional Details of Experiments on the GFootball Environment

The GFootball environment returns a unit reward for each goal scored. For our approach, we consider an agent controlling all 11 players; in contrast, the implementations of some baselines we compare to only control one player. The task where the agent must control all 11 players is significantly more challenging, and we are unaware of any 11-player baseline. This is because, our action space is much larger, and difficult for exploration.

Similar to fetch and stack, since the environment does not have any teleportation support, we implement option templates using simple planners and open-loop controllers (described below). We also provide the numerical values plotted in Figure 6 in Table 9.

A.4.1 Additional Environment and Option Template Details

The dimensions of the football field is bounded by the following limits : $[-1, 1]$ along the x-coordinate, and $[-0.42, 0.42]$ along the y-coordinate. Of the 19 available actions, there are 8 movement actions, one for each of the following directions - {top, top-right, right, bottom-right, bottom, bottom-left, left, and top-left }. There are three actions for passing - { long pass, short pass and high pass}, and one for shooting. There are actions to toggle the following modes - { sprinting, dribbling and movement}. The environment permits an action to let the game engine pick a default move.

The implementations of the option templates in Table 4 are described below :

Charge to the opponent's goal: This option template makes the ball controlling player on the agent's team run in a straight line to the center of the opponent's goal. The precondition for this option template being that the agent's team has the ball. The option template terminates when a player with the ball is within a distance of 0.3 units from the goal (1,0). If the precondition is satisfied, the player sprints in one of the above 8 directions, depending on the angle between the ball controlling player and the opponent's goal.

Maintain ball possession: This option template enables the agent's team to maintain ball possession by either passing the ball to a teammate or move slowly towards the goal while avoiding opponent players who can intercept the ball. When no opponent interceptors are identified ahead of the ball controlling player, in one of 5 relevant directions (top, top-right, right, bottom-right or bottom), the player moves in the identified free direction. Which by default, is towards the opponent's side of the field. This attacker is supported by two players (wings). Otherwise, if no free space is identified, the ball controlling player passes the ball to a teammate with the least number of opponents around him who can intercept the pass. The choice of action (short, long, and high - pass) is based on the distance between the two players.

Shoot: This option template is the same as the environment action to shoot a goal.

Attack and score goals: This option template is a composition of *charge to the opponent's goal*, *maintain ball possession* and *shoot*. When the x coordinate of the ball controlling player is greater than or equal to 0.5, the ball controlling player charges to the goal. When this attacking player is within 0.25 units from the goal, the player takes a shot. Otherwise, the agent's team simply maintains ball possession.

Defend: This option template replicates the inbuilt game engine to block goal attempts by the other team and get ball possession.

All of the above option templates will also terminate after 200 steps. Additionally, for level 1, we mask all but the one hot encoding of ball ownership in the input. Further, while it is possible to combine *defend* with *attack and score goals* to play the game, they are unable to achieve similar performance levels as compared to the baseline or option templates.

A.4.2 Hyperparameters

The Adam optimizer was used to train all networks. The batch size was always set to 40% of the memory size. The exploration (ϵ) is exponentially reduced. All other hyperparameters are given in Table 10. The hyperparameters for level 2 in Table 10 were also utilized in the baseline.

Parameters	level 1			level 2		
	easy	medium	hard	easy	medium	hard
discount factor (γ)		0.9		0.93	0.93	0.96
memory size (transitions)		6000		6000	3000	3000
learning frequency		every 3000 steps		every 3000 steps	every 30,000 steps	
learning rate		0.001		0.001	0.01	0.005

Table 10: Hyperparameters for option templates in gfootball environment.