Hierarchical Behaviour Spaces

Michael Matthews, Anssi Kanervisto, Jakob Foerster, Pierluca D'Oro, Scott Fujimoto, Mikael Henaff

Keywords: Hierarchical RL, Exploration

Summary

Recent work in hierarchical reinforcement learning has shown success in scaling to billions of timesteps when learning over a set of predefined option reward functions. We show that, instead of using a single reward function per option, the reward functions can be effectively used to induce a space of behaviours, by letting the controller specify linear combinations over reward functions, allowing a more expressive set of policies to be represented. We call this method Hierarchical Behaviour Spaces (HBS). We evaluate HBS on the NetHack Learning Environment, demonstrating strong performance. We conduct a series of experiments and determine that, perhaps going against conventional wisdom, the benefits of hierarchy in our method come from increased exploration rather than long term reasoning.

Contribution(s)

1. We introduce Hierarchical Behaviour Spaces (HBS), a scalable hierarchical algorithm in which the controller commands options by specifying the coefficients of a linear combination over predefined reward functions.
   Context: Scalable Option Learning (Henaff et al., 2025) was recently proposed as an approach to online hierarchical reinforcement learning that makes effective use of GPU parallelisation. We extend this approach from considering individual options specified by reward functions to the space of behaviours induced by the linear combination of these reward functions. Linear combinations of option reward functions have been investigated before, most notably in Barreto et al. (2019).
2. We evaluate HBS on the NetHack Learning Environment (Küttler et al., 2020), showing strong performance and outperforming all baselines in the setup we consider. Although measuring progress on the benchmark is somewhat subjective, HBS shows what we believe to be the best known visitation rates of various notable waypoints in the early game, which may be an indicator of progress.
   Context: Despite years of existence, NetHack has proven a remarkably resilient benchmark, with no approach making significant progress (Hambro et al., 2022a; Piterbarg et al., 2023; Klissarov et al., 2023; 2024; Paglieri et al., 2024). Win rate is not a meaningful metric, as to the best of our knowledge no artificial agent has ever beaten the game. There is ongoing debate on how best to measure progress on the benchmark, with the initial proposed metric of in-game score now widely seen as flawed (Hambro et al., 2022b; Paglieri et al., 2024; Matthews et al., 2026).

Hierarchical Behaviour Spaces

Michael Matthews[1,2], **Anssi Kanervisto**[1], **Jakob Foerster**[1,2], **Pierluca D'Oro**[1], **Scott Fujimoto**[1], **Mikael Henaff**[1]
michael.tryfan.matthews@gmail.com

[1]**Meta Superintelligence Labs**
[2]**University of Oxford**

Abstract

Recent work in hierarchical reinforcement learning has shown success in scaling to billions of timesteps when learning over a set of predefined option reward functions. We show that, instead of using a single reward function per option, the reward functions can be effectively used to induce a space of behaviours, by letting the controller specify linear combinations over reward functions, allowing a more expressive set of policies to be represented. We call this method Hierarchical Behaviour Spaces (HBS). We evaluate HBS on the NetHack Learning Environment, demonstrating strong performance. We conduct a series of experiments and determine that, perhaps going against conventional wisdom, the benefits of hierarchy in our method come from increased exploration rather than long term reasoning.

1 Introduction

Learning in long-horizon environments is a central problem in training reinforcement learning (RL) agents. Hierarchical RL, where decision making is done at multiple levels of abstraction, has been proposed as a solution to this issue (Sutton et al., 1999; Klissarov et al., 2025). Intuitively, we as humans do not consider our lives as a long sequence of individual muscle twitches, but employ abstraction and hierarchy when considering long term goals. For instance, when deciding on impactful life decisions like moving country or choosing a career, we would typically consider how it would affect high-level concepts like our quality of life over many years, rather than considering in turn how each day would be affected. Despite this seemingly intuitive mapping to human behaviour, hierarchical RL has shown limited practicality in online settings, where *flat* (i.e. non-hierarchical) methods are still overwhelmingly the tool of choice.

Rather than trying to learn hierarchy end-to-end, recent work (Henaff et al., 2025) has proposed making use of a set of predefined reward functions to guide the learning of low-level policies, with the high level controller policy then learning how to sequence these. While this has shown success, the agent is limited in its expressivity to picking one of the predefined option rewards at any given time.

To help overcome this limitation, we propose allowing the controller to linearly interpolate between the reward functions. This increases the expressivity of low-level policies that can be induced, as even simply interpolating between 2 reward functions can induce arbitrary behaviour that is not present at either extremes. As the dimensionality of the behaviour space increases this effect only grows. We call this method Hierarchical Behaviour Spaces (HBS).

We evaluate our method in the challenging NetHack Learning Environment (NLE) (Küttler et al., 2020): an unsolved benchmark based on the game of NetHack that requires reasoning over very long horizons to solve. We show that HBS outperforms prior work in the NLE and makes material

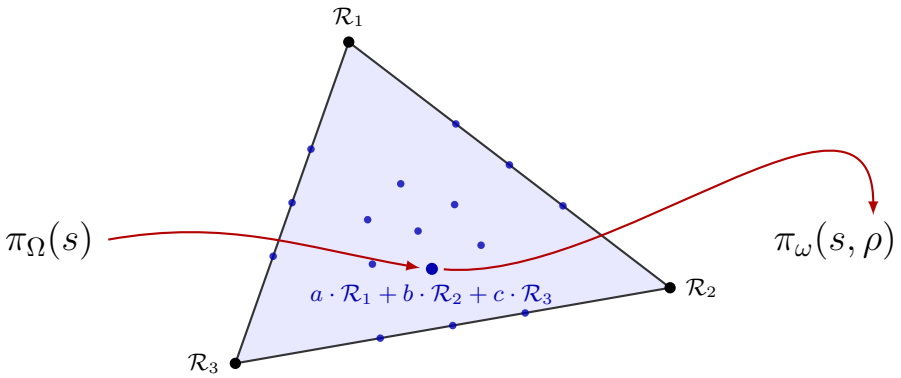

Figure 1: The n reward functions induce a $n-1$ simplex of behaviours. The controller π_Ω selects a linear combination of reward functions from the simplex, which the intra-option policy π_ω then acts to maximise for some number of timesteps. In this way, a large diversity of behaviours can be extracted from only a few reward functions. In practice, we quantise the simplex into a discrete set of linear combinations, shown by the blue dots.

progress on the benchmark, reaching various parts of the game with a consistency higher than any prior work, to the best of our knowledge.

2 Background

2.1 Reinforcement Learning

We consider the standard RL formulation (Sutton et al., 1998), with a Markov Decision Process (MDP) M defined as $M = \langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{T}, \gamma \rangle$, where $\mathcal{S}$ is the set of states; $\mathcal{A}$ is the set of actions; $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition function, $\mathcal{R} : \mathcal{S} \to \mathbb{R}$ is the reward function and γ is the discount factor. At each time step t, a state $s_t \in \mathcal{S}$ is given to the agent who chooses an action a_t in response, causing the environment to transition to a new state $s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)$ and a reward $R_t = \mathcal{R}(s_t, a_t)$ to be given to the agent. The goal of the agent is to learn a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ which maximises its expected discounted sum of rewards $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t R_t]$.

2.2 Options Framework

The options framework (Sutton et al., 1999) extends the classic RL formulation with the notion of temporally extended behaviours or *options*. A policy-over-options or *controller* policy $\pi_\Omega : \mathcal{S} \to \Delta(\Omega)$ selects which option $\omega = (\pi_\omega, \beta_\omega)$ to run from the set of options Ω, surrendering control to the intra-option policy $\pi_\omega : \mathcal{S} \times \Omega \to \Delta(\mathcal{A})$, which executes primitive actions until the termination function $\beta_\omega : \mathcal{S} \times \Omega \to \Delta(\{0, 1\})$ ends the option. After termination, control is passed back to the controller policy, which selects a new option to execute, and the process continues. Some works additionally define initiation sets $\mathcal{I}_\omega$ which restrict the set of states where an option can be initiated, but we consider the case where $\mathcal{I}_\omega = \mathcal{S}$, meaning all options can be initiated from any state.

A set of options Ω induces a semi-MDP (SMDP) on M which we denote M_Ω. A SMDP is analogous to an MDP where options replace actions and the transition function $\mathcal{T}_\Omega : \mathcal{S} \times \Omega \to \Delta(\mathcal{S})$ is given by the probability of reaching s' upon termination when executing option ω starting from s. The reward for a given transition (s, ω, s') is the summed task reward over the course of π_ω's execution starting at s and ending at s'.

2.3 Scalable Option Learning

We focus on the setting proposed in Henaff et al. (2025), where both the controller and intra-option policy are learned concurrently, with each option associated with a reward function $\mathcal{R}^\omega : \mathcal{S} \times \Omega \to \mathbb{R}$. This reward can either be given a priori or learned using some form of reward synthesis, we focus on the former in our experiments. Each intra-option policy π_ω is trained to maximise the cumulative expected return under its reward function $\mathbb{E}_{R_t^\omega \sim \pi_\omega(\cdot,\omega)}[\sum_{t=0}^{\infty} \gamma^t R_t^\omega]$, under the assumption that it will be active for the rest of the episode. In this sense, each option is trained solely to optimise its own reward function in isolation, without regard to other options or the controller policy, in contrast to prior work (Bacon et al., 2017) which optimise both option and controller policies to maximise the task reward alone. The controller policy optimises cumulative expected return over the task reward function $\mathcal{R}_\Omega : \mathcal{S} \to \mathbb{R}$. Rather than learning a termination function β, the controller decides the length for which the option should run for by acting in an augmented option space $|\mathcal{R}| \times |\mathcal{L}|$ where $\mathcal{L}$ is the set of valid option lengths which in practice are exponentially spaced ($\mathcal{L} = \{1, 2, 4, 8, 16..., 128\}$). In the rest of the manuscript we omit the option length action for clarity.

3 Hierarchical Behaviour Spaces

3.1 Motivation

We first discuss some theoretical results from Fruit & Lazaric (2017) that motivate our algorithm.

Theorem 1. *Let M be an MDP, Ω a set of options, and M_Ω the corresponding SMDP. Let π_Ω be any stationary policy on M_Ω and μ the induced low-level policy on M. Let A be any learning algorithm operating in the SMDP M_Ω, let m be the number of option calls executed in M_Ω with execution lengths $l_1, l_2, \ldots, l_m$, and let $T_m = \sum_{i=1}^{m} l_i$ be the corresponding number of steps executed in the original MDP M. The following relationship then holds:*

$$\text{Regret}(M, A, T_m) = \text{Regret}(M_\Omega, A, m) + T_m(V_M^\star - V_{M_\Omega}^\star) \tag{1}$$

where $V_M^\star$ and $V_{M_\Omega}^\star$ denote the maximum value in the original MDP and SMDP respectively.

This result says that when learning a controller in the high-level SMDP, the regret in the original MDP M over a time horizon T_m can be decomposed into the sum of two terms: the regret in the SMDP M_Ω, and the gap between the optimal value in the original MDP and the optimal value in the SMDP. The first term reflects the learning speed in the temporally compressed SMDP, which will typically be faster than in the original MDP due to its shorter horizon ($m \ll T_m$). The second term arises when the set of option policies is not expressive enough to fully represent the optimal policy $\pi^\star$ in the original MDP. That is, if at some state s visited by the agent, there are no option policies π_ω which match $\pi^\star$ for the next few steps, performance may suffer in the original MDP despite the controller being optimal in the SMDP.

This result highlights the need for a set of option policies that is sufficiently expressive to represent the optimal policy as closely as possible. Given a set of n reward functions, where n may be small, SOL (Henaff et al., 2025) will extract n options, which may be insufficiently expressive. We consider instead using these reward functions to induce a *space of behaviours* by considering linear combinations of them, which may be far more expressive.

3.2 Method

We propose Hierarchical Behaviour Spaces (HBS), a two-level hierarchical RL algorithm in which the controller commands behaviours induced from a simplex of predefined reward functions $\{\mathcal{R}_1, ..., \mathcal{R}_n\}$.

HBS jointly trains a controller $\pi_\Omega : \mathcal{S} \to \Delta([0, 1]^n)$ that acts in a temporally compressed process by specifying a vector of reward coefficients $\rho \in [0, 1]^n$, and a behaviour-conditioned policy π_ω :

$\mathcal{S} \times [0,1]^n \to \Delta(\mathcal{A})$ that outputs primitive actions conditioned on the reward coefficients output by the controller (Figure 1).

The two policies act on different time scales and are trained to optimise their respective cumulative discounted return with different discount factors:

$$\pi^*_\Omega = \text{argmax}_{\pi_\Omega} \mathbb{E}_{R_t \sim \pi_\Omega} \left[\sum_{t=0}^{\infty} \gamma_\Omega^t R_t \right] \tag{2}$$

$$\pi^*_\omega(\cdot, \rho) = \text{argmax}_{\pi_\omega} \mathbb{E}_{R_t \sim \pi_\omega(\cdot,\rho)} \left[\sum_{t=0}^{\infty} \gamma_\omega^t \cdot \rho^T(\mathcal{R}_t^1, \ldots, \mathcal{R}_t^n) \right] \tag{3}$$

Where $\rho^T(\mathcal{R}_t^1, \ldots, \mathcal{R}_t^n)$ is the linear combination of the reward functions weighted by the behaviour vector ρ.

HBS can be seen as a generalisation of SOL to allow for interpolation between the various reward functions, rather than simply having to pick one. This greatly expands the range of possible behaviours that the controller can induce, allowing us to reduce the second term in Equation 1.

From a practical perspective, this entirely changes how we should think about the set of reward functions. In SOL each reward function $\mathcal{R}_i$ can be paired with a policy π_i that maximises it, with these then being sequenced by the controller. Each π_i should be a self-contained sub-policy that is optimal for given sub-trajectories. In contrast, the reward functions in HBS define *axes of behaviour*. There is no requirement for any one reward function to induce a sensible policy by itself, rather each reward function should ideally increase the set of representable behaviours the controller can induce, by being orthogonal to the other reward functions in behaviour space.

As with SOL, both the controller and intra-option policies are implemented as separate heads on top of a shared network. We kept the other design choices which enable high throughput, such as parallelised advantage and return computations, allowing us to scale this method to run for billions of timesteps.

In practice, we found that using discrete, quantised bins for the controller to specify the coefficients outperformed using a continuous action space, even though it reduces the expressivity of policies that can be induced, in line with prior work (Farebrother et al., 2024). We also found that *not* normalising the reward coefficients by the sum of the magnitudes marginally improved performance, even though this means that redundant behaviour vectors can be specified.

4 Experiments

4.1 Experimental Setup

We evaluate our method on the NetHack Learning Environment (NLE, Küttler et al. (2020)): a challenging, unsolved benchmark that requires reasoning over very long time horizons[1]. We implement HBS using the SOL (Henaff et al., 2025) codebase, which is itself based off the PPO (Schulman et al., 2017) implementation from Sample Factory (Petrenko et al., 2020).

We follow Matthews et al. (2026) in our setup, meaning that unlike most prior work on the NLE, we make use of the full action space. This actually somewhat alleviates the need for long term credit assignment, as the full action space allows the agent to induce a limited set of useful macro-actions that can span over many in-game steps. For instance, the agent can (and indeed does learn to) execute the actions `9`, `9`, `s`, which causes the agent to wait and heal for 99 timesteps. Conversely, the agent can also take actions that do not step the in-game turn counter, for instance by navigating around menus. The MDP that the agent acts in does not therefore perfectly align with the underlying

[1]The average successful human ascensions take $10^4 - 10^5$ turns (Hambro et al., 2022b; Paglieri et al., 2024).

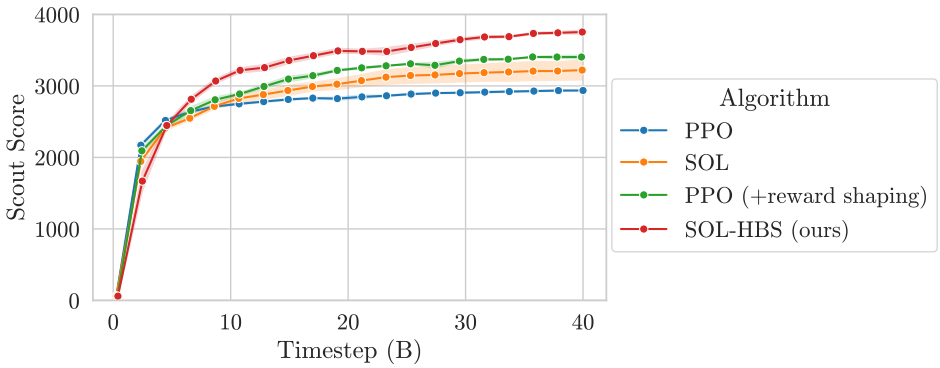

Figure 2: Results on the NetHack Learning Environment. The shaded areas denote 1 standard error over 5 seeds. We show the best setting for each method: for HBS this is using the full space of 5 reward functions, while for SOL and PPO with reward shaping this is just using the scout and $-\boldsymbol{\Delta}(\textbf{dlvl})$ rewards.

game and we make a distinction between *timesteps* (measure of time in MDP) and *turns* (measure of time in NetHack).

In further contrast to prior work on the NLE, we use the *scout* rather than *score* as the reward function we seek to maximise, as recommended by Matthews et al. (2026). As has been noted many times in prior work (Küttler et al., 2020; Hambro et al., 2022a;b; Wołczyk et al., 2024; Paglieri et al., 2024; Matthews et al., 2026), the in-game score is a flawed metric for assessing progress on the NLE, as it is easily gameable by staying on the first dungeon level and killing weak enemies as they spawn, without making any actual progress through the game. Scout reward is increased by revealing new parts of the dungeon map. Since the in-game map is finite, maximising scout reward will lead the agent to the final level, whereas score is unbounded and can be infinitely maximised while never leaving the first level.

Along with the scout reward, we choose 4 other reward functions to define the behaviour space:

- $-\boldsymbol{\Delta}(\textbf{dlvl})$: A reward for decreasing the agent's dungeon level. While progress is generally made by increasing the dungeon level, it is often useful to return to prior levels, for example to retreat or navigate between dungeon branches.
- $-\boldsymbol{\Delta}(\textbf{AC})$: A reward for decreasing armour class, corresponding to better armour.
- $+\boldsymbol{\Delta}(\textbf{Food})$ A reward for eating food. Hunger increases every turn and starvation is a common mode of death.
- $+\boldsymbol{\Delta}(\textbf{XP})$ A reward for increasing experience, which is mostly gained by killing enemies.

These rewards form a 5-dimensional behaviour space for HBS to operate in. We also consider using these rewards directly as options for SOL, as well as simply adding them to PPO as intrinsic rewards. As with the evaluation in Henaff et al. (2025), we are unable to practically compare to other hierarchical methods, as we cannot feasibly run them for the billions of timesteps required by the NLE. We detail the hyperparameters used in Appendix A.

4.2 Results

The results in the NLE are shown in Figure 2 where we run for 40 billion timesteps. We see that, while HBS is less sample efficient at first, it begins outperforming the baselines at around 5 billion timesteps and converges to a higher return. Notably, this is in contrast to prior work in HRL which has posited sample efficiency as a benefit of hierarchy (Klissarov et al., 2025).

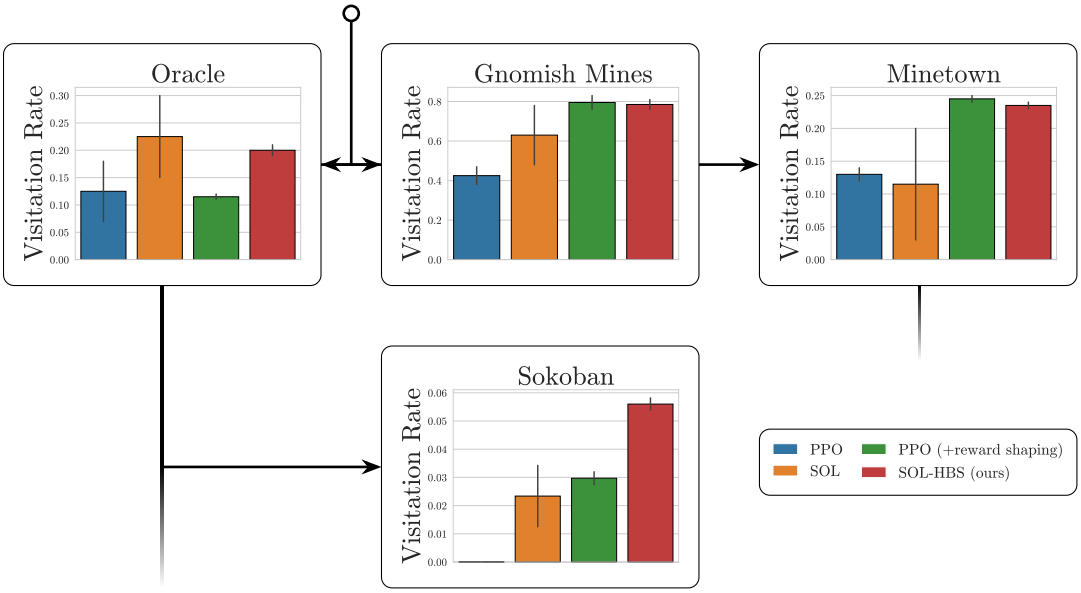

Figure 3: Visitation rates of early waypoints in NetHack for each method. The arrows indicate the underlying structure of the waypoint locations in the game. The error bars show 1 standard error over 5 seeds. HBS achieves high visitation rates on both branches of the dungeon, unlike PPO which tends to favour the Gnomish Mines and SOL which, depending on the seed, will favour one of the branches but not generally learn to navigate between them.

Taking a closer look at individual milestones in the NLE (Figure 3), we can see the qualitative differences between the methods. The Gnomish Mines form the first branch off from the main dungeon in NetHack, with the entrance somewhere on dungeon levels 2–4. We see that vanilla PPO enters them on less than half of all episodes. This is likely due to it implementing a strategy to always descend and never revisit previous dungeon levels or different branches. This means that if the agent descends past the entrance to the mines by continuing down the main dungeon, then it will never go back to find it. In contrast, we see that SOL and especially HBS and PPO with the $-\boldsymbol{\Delta}(\mathbf{dlvl})$ intrinsic reward enter the mines far more regularly, indicating that those agents have learned to traverse back to previous dungeon levels.

The Oracle appears between levels 5 and 9 in the main branch. We see both variants of PPO visit it quite rarely, while SOL visits it frequently on some seeds but infrequently on others. HBS is the only agent to show strong visitation for both branches of the dungeon.

We also see that all agents except vanilla PPO start making progress on finding the Sokoban branch of the dungeon, which appears one level beneath the Oracle, with HBS finding it roughly twice as often as the next best agent. As well as simply requiring the agent to survive for many more floors, this branch is entered on an up-stair and therefore requires the agent to have discovered that going back up the dungeon can sometimes be beneficial.

5 Discussion

5.1 Does HBS allow reasoning over long horizons?

Long term reasoning is often invoked as a motivation behind HRL methods, but it is not clear for our case, whether this is backed up by evidence.

First off we note that increasing the discount factor γ as high as 0.9995 for a simple PPO agent on the NLE produces monotonically increasing returns, before performance collapses at higher γ,

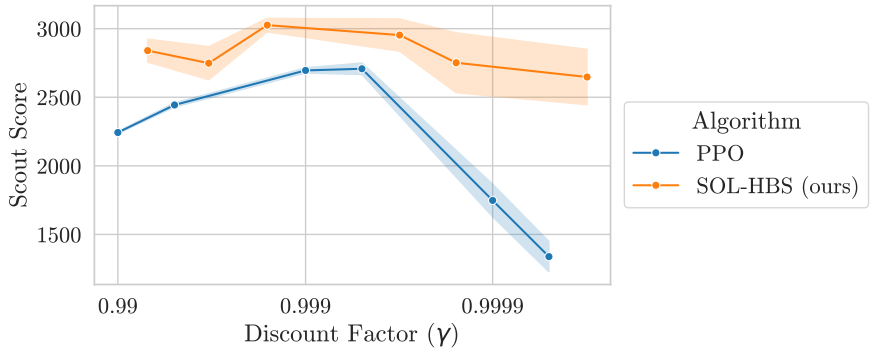

Figure 4: Results on the NetHack Learning Environment for PPO and HBS trained for 8 billion timesteps with different discount factors. The shaded area denotes 1 standard error over 5 seeds. For HBS we vary the controller discount factor γ_Ω, while keeping the intra-option discount factor γ_ω fixed at 0.999. PPO performs well at intermediate values but performance collapses with high discount factors. HBS is largely unaffected by the discount factor of the controller.

likely due to the increased variance of return estimation. This indicates that the ability to act with respect to long term rewards is important in the NLE, as one would intuitively expect.

We then consider HBS, where we fix the intra-option discount factor $\gamma_\omega = 0.999$ but vary the controller discount factor γ_Ω. Note that γ_Ω corresponds to discounting in the primitive MDP, so it can be directly compared to γ_ω and γ. We see that γ_Ω has a limited effect on the performance, and perhaps surprisingly, peaks at a similar point to γ for PPO (Figure 4).

This result challenges the intuition that the benefits of HBS come from the ability for the controller to reason over longer timescales, as we see that the best performance comes roughly when $\gamma_\Omega = \gamma_\omega$ (note we are not claiming this is a general rule, it is just what we observe in this limited experiment). Interestingly, HBS with a low γ_Ω close to 0.99 still outperforms PPO with a γ of 0.9995.

This raises the obvious question: if HBS does not improve long term credit assignment, then why does it outperform other methods?

5.2 HBS as automated tuning of intrinsic rewards

An alternative way to view HBS is as a method for automatic tuning of intrinsic rewards. Consider simply using our predefined set of rewards as intrinsic bonuses for a flat agent, as we did for a baseline. Performing a hyperparameter sweep over every combination of coefficients scales exponentially with the number of reward functions and quickly becomes intractable. HBS can be seen as a method for automatically finding the best coefficients for each intrinsic reward.

But HBS doesn't simply set the coefficients once, it rather dynamically modifies them throughout the episode. This could be seen as expanding the expressivity of intrinsic rewards from static to dynamic bonuses, in effect massively increasing the set of intrinsic reward functions we are optimising over.

This alleviates much of the burden from the RL practitioner, who can now simply specify a set of intrinsic rewards that might be useful (and maybe even only some of the time) and let HBS figure out which ones to apply and when. A key indicator of whether this claim is manifested in reality is how performance is impacted as we add more intrinsic rewards to the mix. Figure 5 indeed shows that as we add to the set of reward functions, HBS performance does generally increase, in contrast to SOL where performance decreases.

This implies that the performance improvements from the increased expressivity of HBS could come from enhanced exploration rather than any benefits to long term credit assignment.

Figure 5: Performance on the NLE as the option space grows for both SOL and HBS at 40 billion timesteps. The shaded area denotes 1 standard error over 5 seeds. The agents start with just the scout and $-\boldsymbol{\Delta}(\textbf{dlvl})$ rewards, before $+\boldsymbol{\Delta}(\textbf{Food})$, $-\boldsymbol{\Delta}(\textbf{AC})$ and $+\boldsymbol{\Delta}(\textbf{XP})$ are added in that order. We see that HBS can make use of the extra axes of behaviour given by the new reward functions, in contrast to SOL whose performance degrades with more options.

6 Related Work

6.1 Hierarchical RL

While early work in Hierarchical RL (Dayan & Hinton, 1992; Kaelbling, 1993; Sutton et al., 1999; Precup, 2000) often focused on predefined options (Kaelbling, 1993; Sutton et al., 1999; Precup, 2000), the trend in later work has been to try and learn options end-to-end in service of a single task reward either through subgoals (McGovern & Barto, 2001; Stolle & Precup, 2002; Menache et al., 2002), arbitrary learned options (Bacon et al., 2017) or options that themselves directly optimise the task reward (Klissarov et al., 2017; Li et al., 2019; Klissarov & Precup, 2021). Similar work has looked at extracting a diverse set of options from gathered data (Gregor et al., 2016; Eysenbach et al., 2018; Sharma et al., 2019) and from offline trajectories (Pertsch et al., 2021; Shi et al., 2022; Park et al., 2023; 2025). We consider the question of reward synthesis for options to be an important but orthogonal line of investigation to our work, where we focus solely on learning the best possible agent given a set of reward functions.

Most similar to our work is the Option Keyboard (Barreto et al., 2019), which learns a hierarchical policy over linear combinations of "cumulants". The Option Keyboard employs a two stage learning process, where value functions are first learned independently and linear combinations can then be synthesised zero-shot. We differ in adopting a jointly trained, coefficient-conditioned architecture that scales to environments like the NLE.

6.2 NLE

While much of the initial work on the NLE was done with tabula rasa RL (Küttler et al., 2020; Hambro et al., 2022a), a wide range of methods have been applied since. These include using offline data (Hambro et al., 2022b; Piterbarg et al., 2023; Wołczyk et al., 2024), LLM generated rewards (Klissarov et al., 2023; 2024; Zheng et al., 2024), LLM actors (Paglieri et al., 2024), symbolic agents (Hambro et al., 2022a), exploration bonuses (Henaff et al., 2022) and hierarchical agents (Matthews et al., 2022; Klissarov et al., 2024; Henaff et al., 2025). As of this date and to the best of our knowledge, humans remain the only agents to have ever beaten the game.

7 Conclusion

In conclusion, we present HBS, a hierarchical RL algorithm in which a controller commands a linear combination of reward functions for an intra-option policy to follow. We show that this setup allows us to train a hierarchical agent with a greatly increased expressivity compared to prior work. We show state of the art results on the NLE, a challenging, unsolved benchmark that requires significant exploration. We hope that HBS can serve as a useful tool for RL practitioners working in long-horizon and hard exploration environments.

References

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

André Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, Daniel Toyama, Shibl Mourad, David Silver, Doina Precup, et al. The option keyboard: Combining skills in reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. *Advances in neural information processing systems*, 5, 1992.

Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz, and Sergey Levine. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Jesse Farebrother, Jordi Orbay, Quan Vuong, Adrien Ali Taïga, Yevgen Chebotar, Ted Xiao, Alex Irpan, Sergey Levine, Pablo Samuel Castro, Aleksandra Faust, et al. Stop regressing: Training value functions via classification for scalable deep rl, 2024. *URL https://arxiv. org/abs/2403.03950*, 2024.

Ronan Fruit and Alessandro Lazaric. Exploration-Exploitation in MDPs with Options. In Aarti Singh and Jerry Zhu (eds.), *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pp. 576–584. PMLR, 20–22 Apr 2017.

Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. *arXiv preprint arXiv:1611.07507*, 2016.

Eric Hambro, Sharada Mohanty, Dmitrii Babaev, Minwoo Byeon, Dipam Chakraborty, Edward Grefenstette, Minqi Jiang, Jo Daejin, Anssi Kanervisto, Jongmin Kim, et al. Insights from the neurips 2021 nethack challenge. In *NeurIPS 2021 Competitions and Demonstrations Track*, pp. 41–52. PMLR, 2022a.

Eric Hambro, Roberta Raileanu, Danielle Rothermel, Vegard Mella, Tim Rocktäschel, Heinrich Küttler, and Naila Murray. Dungeons and data: A large-scale nethack dataset. *Advances in Neural Information Processing Systems*, 35:24864–24878, 2022b.

Mikael Henaff, Roberta Raileanu, Minqi Jiang, and Tim Rocktäschel. Exploration via elliptical episodic bonuses. *Advances in Neural Information Processing Systems*, 35:37631–37646, 2022.

Mikael Henaff, Scott Fujimoto, Michael Matthews, and Michael Rabbat. Scalable option learning in high-throughput environments. *arXiv preprint arXiv:2509.00338*, 2025.

Leslie Pack Kaelbling. Learning to achieve goals. In *IJCAI*, volume 2, pp. 1094–8, 1993.

Martin Klissarov and Doina Precup. Flexible option learning. *Advances in Neural Information Processing Systems*, 34:4632–4646, 2021.

Martin Klissarov, Pierre-Luc Bacon, Jean Harb, and Doina Precup. Learnings options end-to-end for continuous action tasks. *arXiv preprint arXiv:1712.00004*, 2017.

Martin Klissarov, Pierluca D'Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal Vincent, Amy Zhang, and Mikael Henaff. Motif: Intrinsic motivation from artificial intelligence feedback. *arXiv preprint arXiv:2310.00166*, 2023.

Martin Klissarov, Mikael Henaff, Roberta Raileanu, Shagun Sodhani, Pascal Vincent, Amy Zhang, Pierre-Luc Bacon, Doina Precup, Marlos C Machado, and Pierluca D'Oro. Maestromotif: Skill design from artificial intelligence feedback. *arXiv preprint arXiv:2412.08542*, 2024.

Martin Klissarov, Akhil Bagaria, Ziyan Luo, George Konidaris, Doina Precup, and Marlos C Machado. Discovering temporal structure: An overview of hierarchical reinforcement learning. *arXiv preprint arXiv:2506.14045*, 2025.

Heinrich Küttler, Nantas Nardelli, Alexander Miller, Roberta Raileanu, Marco Selvatici, Edward Grefenstette, and Tim Rocktäschel. The nethack learning environment. *Advances in Neural Information Processing Systems*, 33:7671–7684, 2020.

Alexander C Li, Carlos Florensa, Ignasi Clavera, and Pieter Abbeel. Sub-policy adaptation for hierarchical reinforcement learning. *arXiv preprint arXiv:1906.05862*, 2019.

Michael Matthews, Mikayel Samvelyan, Jack Parker-Holder, Edward Grefenstette, and Tim Rocktäschel. Hierarchical kickstarting for skill transfer in reinforcement learning. *arXiv preprint arXiv:2207.11584*, 2022.

Michael Matthews, Pierluca D'Oro, Anssi Kanervisto, Scott Fujimoto, Jakob Foerster, and Mikael Henaff. Revisiting the NetHack learning environment. In *ICLR 2026 Blog Track*, 2026. URL https://iclr-blogposts.github.io/2026/blog/2026/revisiting-the-nle/.

Amy McGovern and Andrew G Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. 2001.

Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cut—dynamic discovery of sub-goals in reinforcement learning. In *European conference on machine learning*, pp. 295–306. Springer, 2002.

Davide Paglieri, Bartłomiej Cupiał, Samuel Coward, Ulyana Piterbarg, Maciej Wolczyk, Akbir Khan, Eduardo Pignatelli, Łukasz Kuciński, Lerrel Pinto, Rob Fergus, et al. Balrog: Benchmarking agentic llm and vlm reasoning on games. *arXiv preprint arXiv:2411.13543*, 2024.

Seohong Park, Dibya Ghosh, Benjamin Eysenbach, and Sergey Levine. Hiql: Offline goal-conditioned rl with latent states as actions. *Advances in Neural Information Processing Systems*, 36:34866–34891, 2023.

Seohong Park, Kevin Frans, Deepinder Mann, Benjamin Eysenbach, Aviral Kumar, and Sergey Levine. Horizon reduction makes rl scalable. *arXiv preprint arXiv:2506.04168*, 2025.

Karl Pertsch, Youngwoon Lee, and Joseph Lim. Accelerating reinforcement learning with learned skill priors. In *Conference on robot learning*, pp. 188–204. PMLR, 2021.

Aleksei Petrenko, Zhehui Huang, Tushar Kumar, Gaurav Sukhatme, and Vladlen Koltun. Sample factory: Egocentric 3d control from pixels at 100000 fps with asynchronous reinforcement learning. In *International Conference on Machine Learning*, pp. 7652–7662. PMLR, 2020.

Ulyana Piterbarg, Lerrel Pinto, and Rob Fergus. Nethack is hard to hack. *Advances in Neural Information Processing Systems*, 36:37540–37566, 2023.

Doina Precup. *Temporal abstraction in reinforcement learning*. University of Massachusetts Amherst, 2000.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Archit Sharma, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman. Dynamics-aware unsupervised discovery of skills. *arXiv preprint arXiv:1907.01657*, 2019.

Lucy Xiaoyang Shi, Joseph J Lim, and Youngwoon Lee. Skill-based model-based reinforcement learning. *arXiv preprint arXiv:2207.07560*, 2022.

Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *International Symposium on abstraction, reformulation, and approximation*, pp. 212–223. Springer, 2002.

Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Maciej Wołczyk, Bartłomiej Cupiał, Mateusz Ostaszewski, Michał Bortkiewicz, Michał Zając, Razvan Pascanu, Łukasz Kuciński, and Piotr Miłoś. Fine-tuning reinforcement learning models is secretly a forgetting mitigation problem. *arXiv preprint arXiv:2402.02868*, 2024.

Qinqing Zheng, Mikael Henaff, Amy Zhang, Aditya Grover, and Brandon Amos. Online intrinsic rewards for decision making agents from large language model feedback. *arXiv preprint arXiv:2410.23022*, 2024.

Supplementary Materials

The following content was not necessarily subject to peer review.

A Hyperparameters

For the base PPO implementation we use the same hyperparameters as SOL, except we increase γ to 0.999, and modify the number of works and environments, since we use a multi-GPU setup so can accommodate more parallel workers (Table 1).

We also list the additional hyperparameters for SOL (Table 2) and SOL-HBS (Table 3).

We list the hyperparameters for the intrinsic rewards we tried with PPO in Table 4. We tried each intrinsic reward independently, as well as all together at once. Trying every combination would have been $3^4 = 81$ trials (each with 5 seeds, as the NLE has a high variance) and was impractical with our compute budget, especially considering these runs tended to take more than 10 billion timesteps each to converge.

Hyperparameter	Value
Recurrence	256
Normalise Returns	False
V-Trace	True
Num Workers	36
Num Envs Per Worker	32
Batch Size	32768
Reward Scale	0.01
Inventory Encoder	Attention
Inventory Query Heads	4
Discount Factor (γ)	0.999
Value Loss Coefficient	0.5
Exploration Loss Coefficient	0.003
Reward Clip	10000
Max Token Length	12
Model	SymbolicGlyphTokenNetEmbeddingBag
Map Input Type	Glyphs
Inventory Input Type	Tokens
Crop Dimension	12
Epochs	1
Learning Rate	0.0002
Num GPUs	4

Table 1: PPO hyperparameters. These were used for PPO, SOL and SOL-HBS.

Hyperparameter	Value
Controller Exploration Scale	1
Controller Reward Scale	0.001
Num Option Steps	Adaptive: $\{1, 2, 4, 8, 16, 32, 64, 128\}$
Reward Scale $-\boldsymbol{\Delta}$(**dlvl**)	100
Reward Scale $-\boldsymbol{\Delta}$(**AC**)	250
Reward Scale $+\boldsymbol{\Delta}$(**Food**)	0.1
Reward Scale $+\boldsymbol{\Delta}$(**XP**)	4

Table 2: SOL additional hyperparameters. These were used for both SOL and SOL-HBS.

Hyperparameter	Value
HBS Coefficient Spacing	Linear
HBS Num Coefficients	3
HBS Normalise Coefficients	False

Table 3: SOL-HBS additional hyperparameters.

Hyperparameter	Considered Values	Value
Reward Scale $-\boldsymbol{\Delta}$(**dlvl**)	$\{0, 50, 100\}$	100
Reward Scale $-\boldsymbol{\Delta}$(**AC**)	$\{0, 125, 250\}$	0
Reward Scale $+\boldsymbol{\Delta}$(**Food**)	$\{0, 0.05, 0.1\}$	0.1
Reward Scale $+\boldsymbol{\Delta}$(**XP**)	$\{0, 2, 4\}$	0

Table 4: PPO Intrinsic Rewards.