

## A DECLARATION ON LARGE LANGUAGE MODEL USAGE

When writing this paper, we used large language models (ChatGPT and DeepSeek) to polish the grammar and writing. In addition, we employed these models to assist with literature search. Specifically, we leveraged DeepSeek’s search capabilities to identify prior research that integrates curriculum learning with continual learning (Tee & Zhang, 2023; Zhou et al., 2023).

## B DETAILS ON QUADRUPEDAL ROBOT EXPERIMENTS

**The Seven Tasks Included.** Fig. S1 a, b, c, d and e, demonstrate the flat ground, hill (climbing), hill (descending), stairs (climbing) and stairs (descending) terrains. On those terrains, the quadrupedal robots were initialized at the middle of the terrain and were commanded to move in random directions with random angular velocity commands. Moreover, in Fig. S1 f, g, h and i, the gap crossing, box climbing, the hurdle and tilted ramp terrains are demonstrated. On those terrains, the quadrupedal robots were initialized on the platforms located in front of the parkour terrains. The robots were commanded to move forward across the parkour terrains. As demonstrated in Fig. S1 j, in the second sub-stage of the walking training stage, we also modified the walking terrains to include gaps so that the gaze of the robot could be pretrained for performing parkour. Here, we only demonstrate the modified version of the stair terrain, but identical gaps were also added on flat and hill terrains.

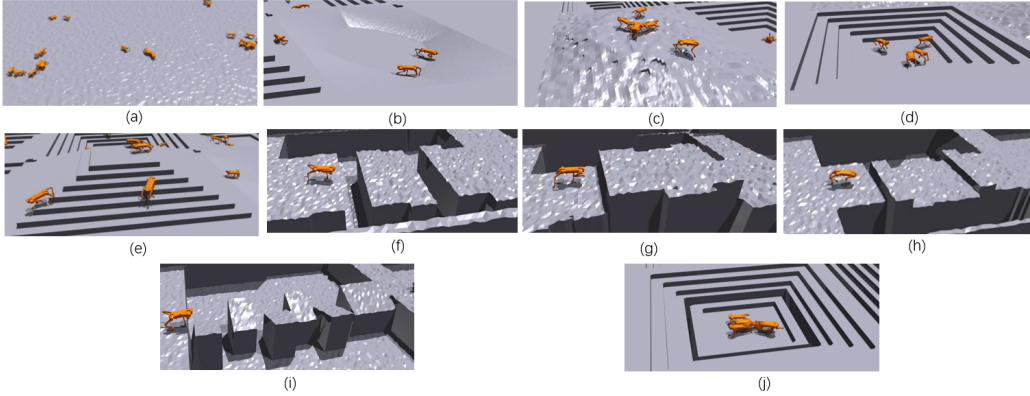


Figure S1: Different terrains included in the quadrupedal robot experiments. (a), (b), (c), (d) and (e) illustrate the flat ground, hill climbing, hill descending, and stairs terrains, stairs climbing and stairs descending terrains. (f), (g), (h) and (i) illustrate the gap crossing, box climb, hurdle and tilted ramp terrains. (h) illustrates the stair terrain added with gaps.

**Training Configuration Details.** In practice, we further divided the two training stages, each stage containing two sub-stages. In the first sub-stage of the first stage, we trained for all three walking tasks and in the second sub-stage, we continued to train for the walking tasks but leveraged the terrains with gaps added (illustrated in Fig. S1h) to pre-train for parkour. In the second stage, we included gap crossing and box climbing tasks in the first sub-stage. In this sub-stage, 50% of terrains were set to gap crossing and 50% of terrains were set to climbing. In the second sub-stage, we further added training for hurdle and tilted ramps, where we set 25% for each type of terrain. Within the second stage, we also included weight decay to increase network plasticity (Dohare et al., 2024).

**Network Architecture for Quadrupedal Robot Policy Network.** In the quadrupedal robot experiments, we leveraged the PPO algorithm to perform reinforcement learning. Within the PPO algorithm, an actor network and a critic network is usually included. Fig. S2 illustrates the design of the actor network we used in our experiments, where it adopted an MLP architecture with two stages. The first stage contains two modules, the first module was a scandot (vision) encoder, which took in five sets of historical and one set of present scandots of the terrain in front of the robot to encode scandots into a feature vector. The historical scandots were taken with an interval of 6 steps relative to the present scandot. This module had two hidden layers with hidden dimensions of 64 and 20. The second module was a past action encoder. It took 8 sets of historical joint positions of the robot (each with 16 values) and encoded them into a past actions feature vector with 32 dimensions.

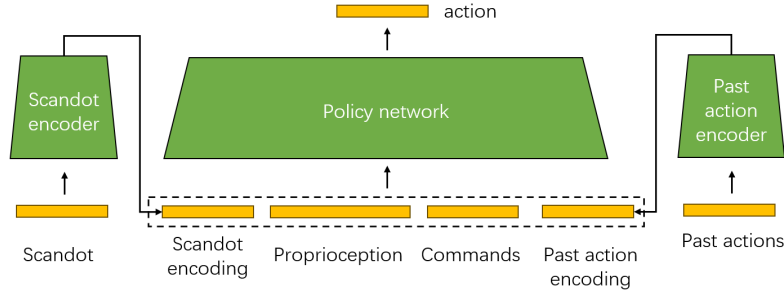


Figure S2: The MLP-based actor network for the policy of the quadrupedal robot.

The encoding was performed with an MLP with a single hidden layer, where the hidden dimension was 64. The second stage was the policy network, where it took in the scandot encoding, direction commands, the past actions encoding and the inertial measurement unit (IMU) information. With the inputs provided, the second block could predict an action. The policy network had 5 hidden layers, with hidden dimensions of 402, 280, 295, 136 and 94. The critic network used in our research had an identical architecture to the actor network, except it had an output layer for generating value predictions. In our experiments, we only added the auxiliary branches to the policy block of the actor network. In the auxiliary branches, we set processing modules to be MLPs with three layers, where the hidden dimension was identical to the dimension of the feature vector to which it is routed. We used the lightweight gating described in Appendix C, where the gating modules were two-layer MLPs with hidden dimensions of 80 and 40 at different depths.

## C LIGHTWEIGHT AUXILIARY BRANCH

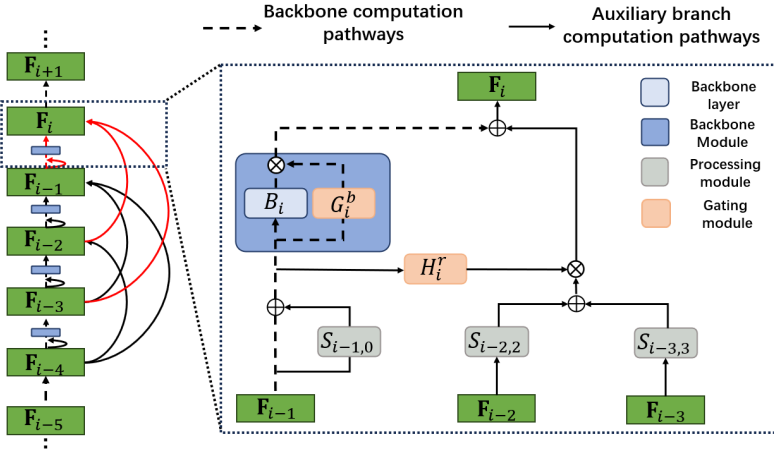


Figure S3: An illustration of applying lightweight auxiliary branches (with span  $N = 2$ ) to features  $\mathbf{F}_{i-4}$ ,  $\mathbf{F}_{i-3}$ ,  $\mathbf{F}_{i-2}$ ,  $\mathbf{F}_{i-1}$  and  $\mathbf{F}_i$  of a feed-forward backbone to form a DPNet. The computation pathways involved in computing  $\mathbf{F}_i$  are highlighted in red and the modules involved are visualized in the dotted box.

Considering that the auxiliary branch design described in the main paper might be computationally prohibitive for some tasks, we also proposed a lightweight version. That is, instead of assigning a gating module for each  $S_{i,m}$ , we assigned a gating module  $H_i^r$  for all shortcut auxiliary branch outputs mapped to each backbone layer. Specifically, as demonstrated in Fig. S3,  $H_i^r$  takes the same input as its corresponding backbone layer and modulates all shortcut module outputs mapped to the current backbone layer simultaneously. Formally, we let the input to  $H_i^r$  be  $\mathbf{F}_{i-1}$  modulated by  $S_{i-1,0}$ :

$$\tilde{\mathbf{F}}_{i-1} = S_{i-1,0}(\mathbf{F}_{i-1}) + \mathbf{F}_{i-1} \quad (8)$$

and we let  $\mathbf{M}_i^r$ , the shortcut auxiliary branch output modulated by  $H_i^r$  be

$$\mathbf{M}_i^r = \sum_{n=1}^N S_{i-n,n}(\mathbf{F}_{i-n}) \odot H_i^r(\tilde{\mathbf{F}}_{i-1}). \quad (9)$$

The computation for obtaining  $\mathbf{M}_i^b$  remains unchanged, but to obtain  $F_i$ , we perform the computation of

$$\mathbf{F}_i = \mathbf{M}_i^b + \mathbf{M}_i^r. \quad (10)$$

With this design, although we have reduced the number of gating modules, but with remaining ones, different pathways with different depths can still be selected, making the policy network a dynamic policy network.

## D MINIHACK TASKS AND EXPERIMENT DETAILS

**The Ten Tasks Included.** We selected 10 tasks from the MiniHack game environment as training tasks, including 1.Room-Random-5x5-v0, 2.Corridor-R2-v0, 3.Room-Dark-5x5-v0, 4.Corridor-R3-v0, 5.Room-Monster-5x5-v0, 6.CorridorBattle-v0, 7.Room-Trap-5x5-v0, 8.HideNSeek-v0, 9.Room-Ultimate-5x5-v0, 10.HideNSeek-Lava-v0. We present examples of the initial observations that an agent may encounter for each task in the MiniHack benchmark in Fig. S4 and the detailed descriptions on the tasks follow below.

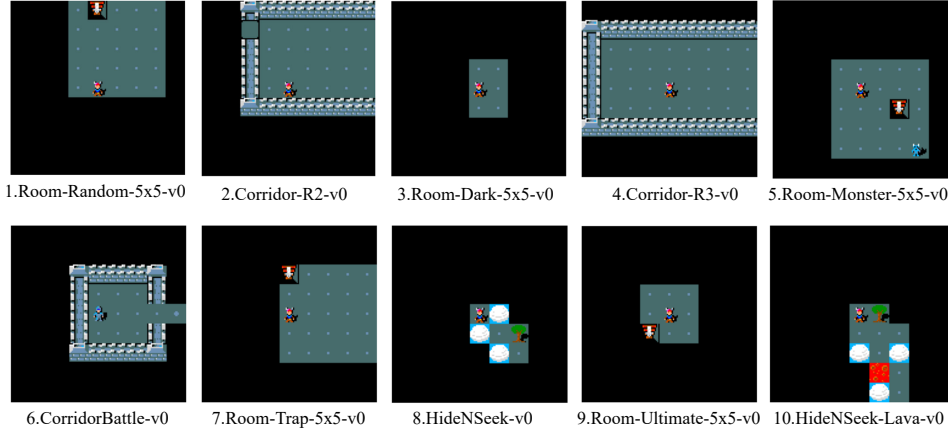


Figure S4: Examples of initial observations for each task in the 10-task MiniHack sequence.

**1.Room-Random-5x5-v0:** Explore the randomly generated room to reach the goal. The layout, player, and goal positions are random in each episode.

**2.Corridor-R2-v0:** Reach the exit by navigating through two connected corridors. The player and exit positions are random in each episode.

**3.Room-Dark-5x5-v0:** Find the goal hidden in the dark room. The player’s position and the goal location are random in each episode.

**4.Corridor-R3-v0:** Reach the exit by navigating through three connected corridors. The player and exit positions are random in each episode.

**5.Room-Monster-5x5-v0:** Reach the goal while avoiding or defeating the monster in the room. Player, monster, and goal positions are random in each episode.

**6.CorridorBattle-v0:** Fight monsters in the corridor and through the corridor to reach the exit. Player, enemies, and exit positions are random in each episode.

**7.Room-Trap-5x5-v0:** Reach the goal while avoiding hidden traps scattered in the room. Player and goal positions are random in each episode.

**8.HideNSeek-v0:** Find and reach the hidden target while avoiding detection. Player and goal positions are random in each episode.

**9.Room-Ultimate-5x5-v0:** Reach the goal while navigating through a room filled with both monsters and traps. Player, monsters, traps, and the goal are random in each episode.

**10.HideNSeek-Lava-v0:** Find and reach the hidden target while avoiding dangerous lava hazards. The target’s position and the lava are random in each episode.

**Training Configuration Details.** In our experiments, we trained all RL agents in the MiniHack environment for 2 epochs using the IMPALA(E. et al., 2018)-based training framework (unless otherwise stated). In each epoch, EWC(J. et al., 2017) was incorporated to facilitate continual learning across all tasks. Each task was trained for 1e6 steps, and during training, all tasks were evaluated every 1e5 steps to record the reward performance. The training hyperparameters of our proposed method on MiniHack are shown in Table S1.

Table S1: The hyperparameters of our proposed method in MiniHack tasks.

Hyperparameters	Ours
Num. actors	16
Learner threads	2
Batch size	25
Unroll length	20
Grad clip	40
Reward clip	[-1,1]
Entropy cost	0.01
Discount factor	0.99
Learning rate	3e-4
EWC $\lambda$	100
EWC, min. task steps	1e6
Fisher samples	100
Normalize Fisher	No
Replay buffer size	1e6

**Network Architecture for MiniHack Policy Network.** For the actor network architecture, we utilized the CNN framework illustrated in Fig. S5. Specifically, we leveraged a CNN backbone consisting of three CNN blocks and two MLP layers, each CNN block was connected with auxiliary branches. For each backbone CNN block, the corresponding processing module within the auxiliary branch contained two convolutional layers, while the gating module comprised two MLP layers followed by a sigmoid activation function. As demonstrated by Fig. S5, the dimension of the MiniHack game image was  $1 \times 3 \times 84 \times 84$ . In the  $\text{CNNBlock}_1$ , we added a maxpooling layer with kernel size 3 and stride 2 between the CNN layers in  $\text{CNNBlock}_1$  to reduce the feature size. The dimensions of hidden features  $F_1, F_2, F_3$  after CNN Blocks were  $1 \times 32 \times 42 \times 42$ . The auxiliary branch pathways we leveraged are illustrated in Fig. 2b. Specifically, the span of the auxiliary branches was 2 and we have only included auxiliary branches for the CNN blocks. After the first MLP layer, the  $F_4$  computed was a vector of dimension 512. The action prediction was obtained after the last MLP layer. The critic network used in our research had an architecture identical to the actor network, except that it had an output layer for generating value predictions.

## E ACTION FREQUENCY SPECTRUM COMPUTATION

To compute the action frequency spectrum of a task, we executed the policy in the environment and collected its output action curves. That is, we collected an action sequence matrix  $\mathbf{M} \in \mathbb{R}^{N \times K \times T}$  where

- $N$  was the number of parallel sequences (samples),
- $K$  was the number of action outputs for the policy network, and
- $T$  was the length (number of time steps) of each sequence.

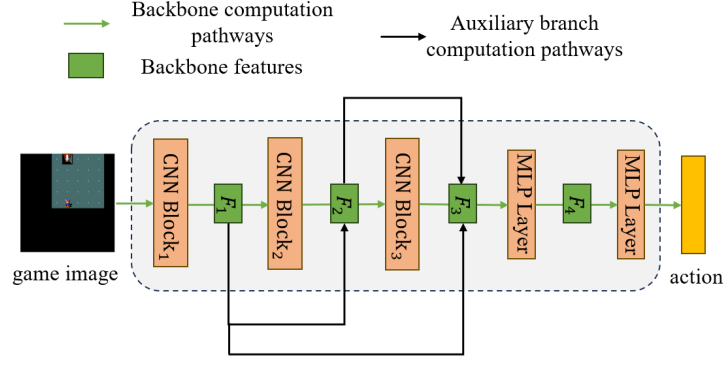


Figure S5: The CNN architecture-based actor network for the policy of the MiniHack tasks.

In our implementation, we performed a real-valued fast Fourier transform (rFFT) along the temporal dimension. To do so, the FFT was computed with a fixed transform length  $n$  (we set  $n = 100$ ) so that the number of frequency components was

$$F = \left\lfloor \frac{n}{2} \right\rfloor + 1. \quad (11)$$

For each sequence  $n \in \{1, \dots, N\}$  and for each action  $k \in \{1, \dots, K\}$ , we defined the discrete Fourier transform (DFT) of the time signal as

$$\hat{M}_{n,k,f} = \sum_{t=0}^{n-1} M_{n,k,t} \exp\left(-\frac{2\pi i f t}{n}\right), \quad \text{for } f = 0, 1, \dots, F-1. \quad (12)$$

Since the input was real-valued, we used the rFFT, which returned only the non-negative frequency components.

Next, the magnitude of the Fourier coefficients was computed as

$$M'_{n,k,f} = \left| \hat{M}_{n,k,f} \right|, \quad (13)$$

which represented the absolute value (i.e., amplitude) of the frequency component  $f$ .

To obtain a robust frequency representation for each action, we averaged the magnitudes over all  $N$  sequences:

$$P_{k,f} = \frac{1}{N} \sum_{n=1}^N M'_{n,k,f}. \quad (14)$$

This yielded the per-action frequency distribution, with

$$\mathbf{P} \in \mathbb{R}^{K \times F}, \quad (15)$$

where the element  $P_{k,f}$  corresponds to the average magnitude of the frequency component  $f$  for action output  $k$  of the policy network.

To compute the overall action frequency spectrum for a task, we need to aggregate across the  $K$  action dimensions by summing over the rows of  $\mathbf{P}$ :

$$d_f = \sum_{k=1}^K P_{k,f}, \quad \text{for } f = 0, 1, \dots, F-1.$$

This resulted in a task-level frequency distribution,

$$\mathbf{d} \in \mathbb{R}^F,$$

where each element  $d_f$  represents the total average amplitude of frequency component  $f$  across all action outputs. This final representation captures the overall frequency characteristics of the policy behavior for the given task type and we plotted it as the action frequency spectrum.

## F COMPUTATION OF EFFECTIVE DECISION DEPTH

This section details the method for computing the *effective decision depth* (EDD) of a forward pass through the Dynamic Policy Network (DPNet). The effective decision depth is a metric that quantifies the cumulative processing depth for an input as it is propagated through the network.

Effective decision depth (EDD) is computed in a single bottom-up sweep through the network. Starting from the input feature  $\mathbf{F}_0$  (with EDD initialized to zero), we proceed layer by layer up to the final feature  $\mathbf{F}_L$ . At each layer  $i$ , we consider all pathways originating from earlier features  $\mathbf{F}_{i-n}$  and compute the depth as the sum of the EDD of the source feature and the EED of the connecting module. If the module is gated, its EED is given by the mean gate activation multiplied by the module’s depth. If the module is not gated, its cost is computed as the mean absolute activation of the module. The EDD of  $\mathbf{F}_i$  is then obtained by averaging the EDD of all incoming pathways. Repeating this process for  $i = 0$  through  $L$  yields the final value  $C_L$ , which represents the overall effective decision depth of the network.

Formally, we let  $C_i$  denote the effective decision depth at the output of the  $i^{\text{th}}$  backbone layer. The computation proceeds as follows:

1. **Initialization:** The effective decision depth for the initial input feature,  $C_0$ , is initialized to 0.
2. **Iterative Update:** For each subsequent backbone layer  $i$ , the effective decision depth  $C_i$  is computed based on the previous  $N$  depths  $\{C_{i-1}, C_{i-2}, \dots, C_{i-N}\}$  and the activations of the layer’s components connected to the current backbone layer output. The update is performed in three steps:

- **Base Depth Contribution:** A base depth value,  $\tilde{C}_i^b$ , is calculated by incorporating contributions from the current backbone layer and its self-dilation auxiliary branch:

$$\tilde{C}_i^b = C_{i-1} + \mathbb{E}[\mathbf{G}_i^b] \cdot D_i^b + \mathbb{E}[|\mathbf{S}_{i,0}|] \cdot D_{i,0}^s \quad (16)$$

where:

- $\mathbb{E}[\mathbf{G}_i^b]$  is the mean output of the backbone gating module at layer  $i$
- $D_i^b = 1$  is the depth of the current backbone layer
- $\mathbb{E}[|\mathbf{S}_{i,0}|]$  is the mean absolute output of the self-dilation processing module
- $D_{i,0}^s$  is the depth of the self-dilation processing module

The products  $\mathbb{E}[\mathbf{G}_i^b] \cdot D_i^b$  and  $\mathbb{E}[|\mathbf{S}_{i,0}|] \cdot D_{i,0}^s$  estimate the effective utilization of the backbone layer and self-dilation processing module, respectively.

- **Skip Connection Contribution:** An average skip contribution  $\tilde{C}_i^s$  is computed by considering influences from  $k$  previous layers (where  $k = \min(i, N)$ ) via shortcut connections:

$$\tilde{C}_i^s = \sum_{j=1}^k (\mathbb{E}[\mathbf{H}_i^r] \cdot D_{i-j,j}^s + C_{i-j}) \quad (17)$$

where  $\mathbb{E}[\mathbf{H}_i^r]$  is the mean output of the gating module governing shortcut auxiliary branch to layer  $i$ , and  $D_{i-j,j}^s$  is the depth of the shortcut auxiliary branch processing module from layer  $i - j$  to layer  $i$ .

- **Final Effective Decision Depth:** The effective decision depth at the current backbone layer output is obtained by averaging the base depth and skip contributions:

$$C_i = \frac{\tilde{C}_i^b + \tilde{C}_i^s}{k + 1} \quad (18)$$

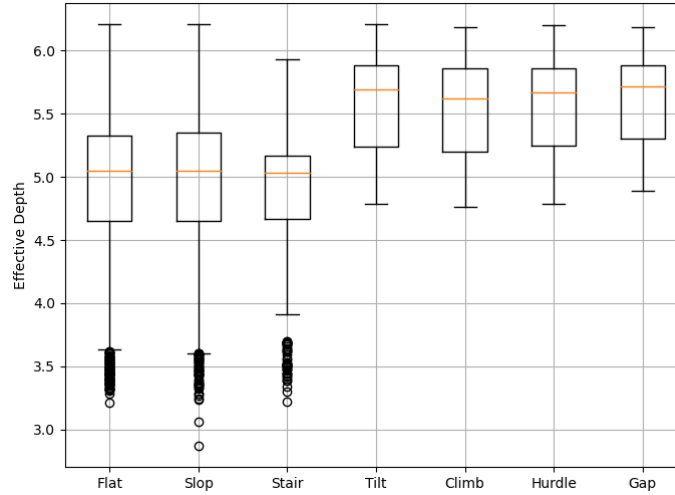


Figure S6: Boxplots for DPNet effective decision depth on the 7 quadrupedal locomotional tasks.

- Termination:** The process repeats iteratively until the effective decision depth  $C_L$  for the final backbone feature  $\mathbf{F}_L$  is computed.

The final value  $C_L$  represents the overall processing complexity for a given input and is reported as the effective decision depth of the forward pass.

## G EFFECTIVE DECISION DEPTH RESULTS ON QUADRUPEDAL LOCOMOTION TASKS

This section compares the effective decision depth of DPNet across different task types, examining the three walking tasks (Flat, Slope, Stair) versus the four parkour tasks (Tilt, Climb, Hurdle, Gap). The analysis reveals distinct depth utilization patterns between the two task categories.

For walking tasks, DPNet consistently employed shallower computational pathways, with median effective depths clustering around 5.0. The lower distribution boundaries were substantially lower, with first quartile values near 4.7 and lower whiskers extending to approximately 3.7. Additionally, all three walking tasks exhibited outliers with effective depths as low as 3.2, indicating instances where minimal computational depth sufficed.

In contrast, parkour tasks demonstrated systematically deeper pathway utilization. Median effective depths were consistently higher (around 5.7), while the first quartile (approximately 5.3) and lower whisker (around 4.8) values were significantly higher compared to walking tasks. This systematic upward shift across all distribution metrics confirms that DPNet adaptively selects deeper computational pathways for compositional parkour tasks while employing shallower pathways for atomic walking tasks.

## H THE FINAL REWARDS OF DIFFERENT MINIHACK TASKS

The final individual MiniHack task rewards for different continual learning baseline algorithms (analyzed in Section 4.4 of the main paper) are presented in Table S3. Our proposed method consistently outperformed the continual learning baseline across most tasks, where the highest rewards reached 1.000. However, on Corridor-R2-V0, CorridorBattle-v0, and HideNSeek-v0, the rewards achieved were slightly lower than those of CLEAR. We attribute this to CLEAR’s use of data replay to maintain individual task performance, an effective but hardware-intensive approach. Overall, our proposed DPNet design and progressive training technique effectively mitigated conflicts, thereby enhanced multi-task reinforcement learning performance on MiniHack.

Table S2: Direct training rewards of different MiniHack tasks.

Task ID	Task name	Direct training rewards
1	Room-Random-5x5-v0	$0.845 \pm 0.003$
2	Corridor-R2-v0	$-0.862 \pm 0.035$
3	Room-Dark-5x5-v0	$0.781 \pm 0.007$
4	Corridor-R3-v0	$-0.748 \pm 0.126$
5	Room-Monster-5x5-v0	$0.677 \pm 0.021$
6	CorridorBattle-v0	$0.018 \pm 0.002$
7	Room-Trap-5x5-v0	$0.816 \pm 0.003$
8	HideNSeek-v0	$0.019 \pm 0.003$
9	Room-Ultimate-5x5-v0	$0.565 \pm 0.023$
10	HideNSeek-Lava-v0	$0.026 \pm 0.002$

Table S3: The final rewards of different individual tasks in MiniHack.

Tasks	EWC (J. et al., 2017)	P&C (S. et al., 2018)	CLEAR (R. et al., 2019)	SANE (S. et al., 2022)	Ours
Room-Random-5x5-v0	$0.897 \pm 0.000$	$0.884 \pm 0.000$	$0.816 \pm 0.003$	$0.819 \pm 0.034$	$1.000 \pm 0.000$
Corridor-R2-v0	$0.049 \pm 0.062$	$0.224 \pm 0.018$	$0.505 \pm 0.049$	$-0.070 \pm 0.140$	$0.480 \pm 0.014$
Room-Dark-5x5-v0	$0.588 \pm 0.000$	$0.850 \pm 0.016$	$0.633 \pm 0.019$	$0.892 \pm 0.023$	$0.744 \pm 0.003$
Corridor-R3-v0	$-0.468 \pm 0.001$	$-0.289 \pm 0.001$	$-0.596 \pm 0.003$	$-0.910 \pm 0.004$	$-0.347 \pm 0.001$
Room-Monster-5x5-v0	$0.931 \pm 0.002$	$0.812 \pm 0.009$	$0.892 \pm 0.008$	$0.963 \pm 0.003$	$1.000 \pm 0.000$
CorridorBattle-v0	$-0.320 \pm 0.001$	$-0.039 \pm 0.000$	$0.357 \pm 0.049$	$0.008 \pm 0.161$	$-0.141 \pm 0.006$
Room-Trap-5x5-v0	$0.863 \pm 0.002$	$0.781 \pm 0.000$	$0.780 \pm 0.032$	$0.999 \pm 0.000$	$1.000 \pm 0.000$
HideNSeek-v0	$0.624 \pm 0.003$	$-0.014 \pm 0.000$	$0.698 \pm 0.027$	$0.731 \pm 0.002$	$0.298 \pm 0.010$
Room-Ultimate-5x5-v0	$0.694 \pm 0.007$	$0.809 \pm 0.002$	$0.636 \pm 0.003$	$0.853 \pm 0.001$	$0.897 \pm 0.010$
HideNSeek-Lava-v0	$0.597 \pm 0.007$	$0.026 \pm 0.002$	$0.664 \pm 0.015$	$0.731 \pm 0.009$	$0.847 \pm 0.002$
Final Average Reward	$0.445 \pm 0.001$	$0.4045 \pm 0.003$	$0.538 \pm 0.005$	$0.502 \pm 0.003$	<b><math>0.578 \pm 0.001</math></b>