PADDLE: Logic Program Guided Policy Reuse in Deep Reinforcement Learning

Hao Zhang[1], **Tianpei Yang**[1,2,*], **Yan Zheng**[1], **Jianye Hao**[1,*]
Matthew E. Taylor[2]
[1]College of Intelligence and Computing, Tianjin University, China
[2]University of Alberta, Canada
{3018216216,tpyang,yanzheng,jianye.hao}@tju.edu.cn
matthew.e.taylor@ualberta.ca

Abstract

Learning new skills through previous experience is common in human life, which is the core idea of Transfer Reinforcement Learning (TRL). This requires the agent to learn *when* and *which* source policy is the best to reuse as the target task's policy, and *how* to reuse the source policy. Most TRL methods learn, transfer, and reuse black-box policies, which is hard to explain 1) when to reuse, 2) which source policy is effective, and 3) reduces transfer efficiency. In this paper, we propose a novel TRL method called **P**rogr**A**m gui**DeD** po**L**icy r**E**use (PADDLE) that can measure the logic similarities between tasks and transfer knowledge with interpretable cause-effect logic to the target task. To achieve this, we first propose a hybrid decision model that synthesizes high-level logic programs and learns low-level DRL policy to learn multiple source tasks. Second, we estimate the logic similarity between the target task and the source tasks and combine it with the low-level policy similarity to select the appropriate source policy as the guiding policy for the target task. Experimental results show that our method can effectively select the appropriate source tasks to guide learning on the target task, outperforming black-box TRL methods.

1 Introduction

Deep Reinforcement Learning (DRL) has achieved success in various domains, including robotics control [29, 1], video games [27, 4], and autonomous driving [31, 23]. However, DRL can suffer from sample inefficiency, making learning from scratch difficult [40, 8]. Transfer Learning (TL) has shown great potential to accelerate DRL by leveraging prior knowledge from past learned tasks [18], i.e., policy reuse in DRL [12, 42, 45].

One important open question in this area is: How can one efficiently reuse source policies to speed up target policy learning? A large number of works have studied the problem, including: (1) Measure the similarity between two tasks by mapping the state space between them [6]; or calculate the similarity of two Markov Decision Processes (MDPs), then transfer value functions directly according to their similarities [34]. However, it is infeasible to extend these methods to complex domains due to the high computational costs. (2) Transfer the source value function to the target task, such as advantage-based experience selection [37, 45]. However, relying too heavily on value functions will result in large value estimation errors when faced with sparse or delayed reward situations [38, 16]. Furthermore, when one source policy is only partially useful for the target task, a single policy measure of similarity is difficult to integrate knowledge from different source tasks, leading to poor transfer. (3) Some

*Corresponding authors: Tianpei Yang (tpyang@tju.edu.cn) and Jianye Hao (jianye.hao@tju.edu.cn).

NeurIPS 2023 Workshop on Generalization in Planning (GenPlan 2023).

works identify a part of useful knowledge from each source policy by adopting a hierarchical high-level policy [25, 42]. However, training additional components will cause additional computational costs, weakening the effectiveness of the transfer. In addition, most existing transfer learning methods mentioned above focus on learning, extracting, and reusing black-box knowledge, which makes it difficult to reveal internal connections between source tasks and target tasks at appropriate granularity. Thus, learning when and which knowledge is effective requires significant learning costs, limits the effectiveness of transfer, and even fails in situations where only slight logical changes occur between different tasks.

This situation can be improved if the transferred knowledge has interpretable cause-effect logic, such as using clearer policy representation to measure semantic correlations between tasks, to easily detect the slight changes in the logic level, and select the appropriate source policies to guide the target task learning. Researchers have used neural-symbolic learning, such as Inductive Logic Programming (ILP) [28], to automatically generate logic programs as the policy for performing logic-driven and interpretable behaviors [11, 22, 7, 17]. Although these methods demonstrate sufficient advantages in logical reasoning, relying on human experts' definitions makes them difficult to apply to complex tasks and limits them to discrete action spaces. To this end, this paper proposes the **P**rogr**A**m gui**D**e**D** po**L**icy r**E**use (PADDLE) algorithm to address the above challenges. PADDLE incorporates a hybrid decision model to learn the policy, which is a two-level model combining the advantages of DRL and program synthesis, where the higher level uses program synthesis to generate logic programs, and the low level adopts arbitrary DRL methods to learn primitive policies. Then PADDLE estimates the logic similarity between each source task and the target task and combines it with the low-level policy similarity to determine which source policy should be reused at different stages (subtask/subgoal). In this way, PADDLE abstracts and aligns the target task's state space and the source tasks' state space at a more granular representation, thus selecting the appropriate source policy in a more effective way. Furthermore, the similarity measurement proposed in this paper is easily computed and relies less on the value function than these advantage-based methods, avoiding the negative influence of value estimation errors. Our contributions can be summarized as follows:

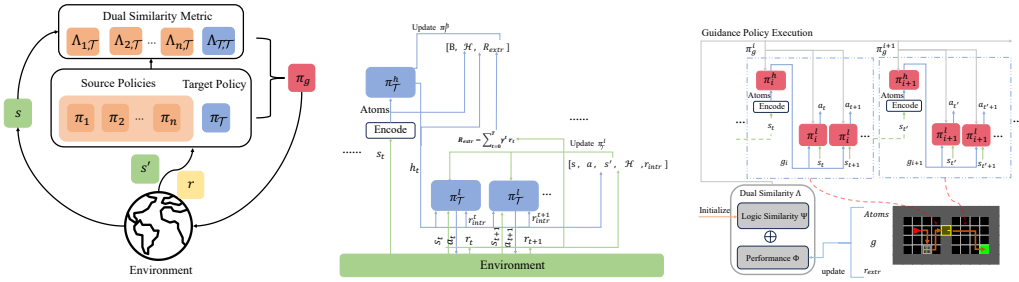
- A hybrid decision model is proposed that demonstrates excellent performance and logical reasoning ability;
- A logic program-based transfer method is proposed that enables efficient knowledge transfer when learning the target task;
- Experimental results that show PADDLE outperforms state-of-the-art transfer baselines on multiple complex tasks in both discrete and continuous domains, exhibiting the advantages of knowledge with interpretable cause-effect logic in TRL.

2 Preliminary

This paper focuses on standard Reinforcement Learning (RL) tasks, where the agents interacting with the environment can be modeled in MDPs. MDPs contain a 5-tuple $\langle S, A, P, R, \gamma \rangle$, with state space S, action space A, transition function $P : S \times A \to S$, reward function $R : S \times A \to \mathbb{R}$, discount factor $\gamma \in [0, 1]$[36]. The goal of RL is to find the optimal policy π^* that is with a state as input and outputs a probability distribution over actions at each time step t so that maximizes the cumulative discounted return: $\pi^* = \arg\max_\pi \mathbb{E}_{a\sim\pi}[\sum_{t\geq 0} \lambda^t r_t]$.

ILP refers to the task of learning a logic program (set of clauses) that extracts a given set of positive examples and does not extract a given set of negative examples [15]. A definite clause can be expressed as: $H :- B_1 \wedge B_2 ... \wedge B_n$, with head atom H and body atoms B_i, $:-$ denotes logical entailment: H is true if $B_1 \wedge B_2 ... \wedge B_n$ is true (especially, logical connectives include negation $\neg$, disjunction $\vee$, etc). An atom is predicate followed by a tuple $p(t_1, ..., t_n)$, where p is a n-ary predicate and $t_i, i \in [1, n]$ are terms, either variables or constants. An atom whose all arguments are constants is called a ground atom and all the concerning ground atoms are called a Herbrand base. A predicate defined based on ground atoms without deductions is called an extensional predicate. Otherwise, it is called an intensional predicate. In ILP tasks, given some initial input predicates (e.g., `has_key(X)`, `is_agent(X)`) and some target predicates (e.g., `gt_key()`, `gt_door`), then combined them into a complete set of clauses according to the pre-defined template, the goal is to find a logic program C that satisfies the pre-defined specification and $C = \text{argmax}_C\, E$, where E is the cumulative discounted return. During each deduction step, C is recursively updated with the

forward chaining mechanism from all positive and negative samples. The attempts that combine ILP with differentiable programming are presented in [30, 11, 22], which our work is based on.



(a) (b) (c)

Figure 1: (a) The overview of PADDLE; π_g represents the guidance policy selected from the policy library, and Λ represents the dual similarity measurement. (b) The hybrid decision-model learning process. (c) The dual similarity and guidance policy selection.

3 Methodology

The PADDLE algorithm is illustrated in Figure 1(a). Initially, a set of source policies are learned using the hybrid decision model, and a target policy is randomly initialized. The goal is to estimate the similarity between each source policy and the target policy to quickly find the appropriate source policies to guide target policy learning. PADDLE comprises two key components: (1) A hybrid decision model, where high-level reasoning can be performed on complex temporal logic problems while using neural networks for low-level exploration; and (2) A transfer algorithm based on the dual similarity measurement Λ, which directly optimizes the target policy by alternatively using knowledge from both the environment and from appropriate source policies. We will provide a detailed introduction to each component of PADDLE in the following sections.

3.1 Hybrid Decision Model

To efficiently synthesize high-level logic programs and corresponding primitive policy, we propose the Hybrid Decision Model (HDM). HDM uses a special hierarchical structure where the high-level policy uses the program synthesis method [47, 22, 30] to synthesize logic programs, while the low-level policy uses a DRL algorithm to learn the primitive policy. The specific learning process is shown in Figure 1(b), where HDM interacts with the environment on different levels of tasks and collects samples to update the corresponding policies. The HDM leverages the ability of program synthesis to reveal the causal logic for a given task and different from the given sequence plan in [21] which has limited generalizability and needs more prior knowledge [17], while also releasing expert knowledge on low-level exploration tasks by lower-cost DRL methods.

We define a symbolic MDP $\langle B, \mathcal{H}, S, P_h, R_h, \gamma_h \rangle$ for the high-level module and a goal-conditioned MDP $\langle S, A, \mathcal{H}, P_l, R_l, \gamma_l \rangle$ for the low-level module. B and $\mathcal{H}$ are a set of pre-defined ground atoms, which describe the environment state (with body atoms) and abstract operators/subgoals (with head atoms), respectively. P_h and P_l represent the transition function, R_h and R_l represent the reward function[2], S and A represent original state space and action space, γ_h and γ_l represent discount factors for the high-level and low-level modules, respectively. At the beginning of training, the high-level module combines body atoms and head atoms to form a complete set of clauses following a specific template (i.e., $H :- B_1 \wedge B_2 ... \wedge B_n$) and attaches weights (initialized to a normal distribution) to each clause to make it differentiable [30]. When interacting with the environment, the high-level module maintains a pre-defined encoder $E(\cdot)$ to map the environment state s to a set of body atoms $(B_1 \wedge B_2 ... \wedge B_n)$, which is used to perform the deduction and activate possible clauses to obtain the probability of subgoal ($[0, 1]^{|\mathcal{H}|}$) (The specific process is not the focus and will not be described [22, 7]). The subgoal selected by the high-level module will not directly affect the environment but

[2]Specifically, R_h is the cumulative external discount reward, R_l is the positive internal reward given after completing the subgoal and the negative internal reward given for not completing the goal.

Algorithm 1 Logic Similarity Function Ψ

for $clause$ in $D_t \in \{D_{\mathcal{S}_1}, ..., D_{\mathcal{S}_N}, D_{\mathcal{T}}\}$ **do**
  if body atom of $clause$ never appeared **then**
    new $c_t^{len(c_t)+1}$ add $clause$
  else
    c_t^k add $clause$ (c_t^k contains $clause$ corresponding body atoms)
  end if
end for
for $c_{\mathcal{T}}^i$ in $c_{\mathcal{T}}$ **do**
  for t in $[\mathcal{S}_1, ..., \mathcal{S}_N, \mathcal{T}]$ **do**
    max_similarity, set_$c = 0$, [ ]
    for c_t^j in c_t **do**
      $\psi(c_{\mathcal{T}}^i, c_t^j) = \gamma_{\text{head}} * \cap_{head(c_{\mathcal{T}}^i), head(c_t^j)} + \gamma_{\text{body}} * \cap_{body(c_{\mathcal{T}}^i), body(c_t^j)}$
      if $\psi(c_{\mathcal{T}}^i, c_t^j) >$ max_similarity **then**
        max_similarity, set_$c = \psi(c_{\mathcal{T}}^i, c_t^j)$, $[c_t^j]$
      end if
      if $\psi(c_{\mathcal{T}}^i, c_t^j) =$ max_similarity **then**
        set_c add c_t^j
      end if
    end for
    $\Psi(c_{\mathcal{T}}^i)$ add (t, max_similarity, set_c)
  end for
end for
Return: Ψ (**Input:** a target task scene $c_{\mathcal{T}}^j$; **Output:** the most similar scenes and corresponding ψ for each source policy)

will serve as a basis for judging whether the low-level module has reached it or the maximum time step. If the corresponding goal is completed, an internal reward r_{intr} (defined in R_l) will be given to the low-level agent. The high-level module records the accumulated real rewards R_{extr} (defined in R_h) during the exploration process.

For policy updates, after each episode, the high-level module collects samples $\{B, \mathcal{H}, R_{extr}\}$ and updates to help maximize the discounted extrinsic reward R_{extr} by training weights of the corresponding clauses with Monte Carlo policy gradient updates [41]:

$$\theta' = \theta + \alpha \nabla_\theta \log \pi_\theta^h Q_{\pi_\theta^h} + \gamma_h \nabla_\theta H\left(\pi_\theta^h\right) \tag{1}$$

π_θ^h is high-level policy (learnable parameters θ) and $H\left(\pi_\theta^h\right)$ is the entropy regularization to improve exploration. Then, according to the trained weights, it synthesizes programs that match the task logic from a complete set of clauses. The low-level module collects samples $\{s, a, s', \mathcal{H}, r_{intr}\}$ (the subgoal $\mathcal{H}$ maps to the state), and updates to maximize the discounted intrinsic reward r_{intr} through the underlying algorithm (DRL method).

3.2 Dual Similarity Measurement

Previous black-box knowledge transfer algorithms lack the ability to detect the similarity behind the task logic, making it difficult to be applied to more complicated domains. Furthermore, measuring the logic similarity among tasks can help explain why and how the transferred knowledge is useful. To achieve this, we propose a dual similarity based on the hybrid decision model.

Given a set of pre-trained source policies, PADDLE compares the set of clauses $D_{\mathcal{S}_i}$ from the logic program synthesized on each source policy with a complete set of clauses $D_{\mathcal{T}}$ from the randomly initialized target policy. In reality, each source policy may only be partially useful for the target task, so it is more practical to measure similarity at the subtask level for reuse. Therefore, for each set of clauses: $D_t \in \{D_{\mathcal{S}_1}, ..., D_{\mathcal{S}_N}, D_{\mathcal{T}}\}$, classify these clauses with the same body atoms into different kinds of scenes $\{c_t^i\}$ (e.g., `[gt_key(), gt_door...]:- ¬has_key(X), is_agent(X), has_key(Y), is_env(Y)`, where c_t^i represents the i-th scene of t task) (Algorithm 1, lines 1–7). For a scene, body atoms and head atoms have different semantic information, and we calculate the

semantic coincidence degree between the head atoms and the body atoms for different tasks as the logic similarity. Obviously, body atoms reflect more environmental context at the subtask level, thus body atoms will be assigned a higher importance weight than head atoms ($\gamma_{\text{body}} > \gamma_{\text{head}}$; detailed analysis in the experiments). For each $c_{\mathcal{T}}^{j}(j \in [1, n_{\mathcal{T}}])$, we get the $c_{\mathcal{S}_i}^{j'}(i \in [1, N], j' \in [1, n_{\mathcal{S}_i}])$ (n_t represents the number of scenes for task t), calculate $\psi(c_{\mathcal{T}}^{j}, c_{\mathcal{S}_i}^{j'})$ (Algorithm 1, lines 12), then use the $c_{\mathcal{S}_i}^{max}$ of maximum $\psi(c_{\mathcal{T}}^{j}, c_{\mathcal{S}_i}^{j'})$ from each source policy to update the logic similarity function Ψ (Algorithm 1, lines 9–21). Therefore, Ψ records the most similar scene and value of each source task for each target task scene. Specifically, Ψ takes as input a target task scene $c_{\mathcal{T}}^{j}$ and outputs the maximum ψ and corresponding $c_{\mathcal{S}_i}^{max}$ from the i-th source task ($i \in [1, N]$). Note that for the i-th source task, there may be multiple $c_{\mathcal{S}_i}^{j'}$ that are equally similar to the current $c_{\mathcal{T}}^{j}$ and each $c_{\mathcal{T}}^{j}$ with its own $\psi(c_{\mathcal{T}}^{j}, c_{\mathcal{T}}^{j}) = Max_{i \in [1,N]} \psi(c_{\mathcal{T}}^{j}, c_{\mathcal{S}_i}^{max})$.

By calculating Ψ, we map each $c_{\mathcal{T}}^{j}$ to $c_{\mathcal{S}_i}^{max}$ in each source task, but this mapping relationship still has defects. When some scenes of the target task do not exist in the source task, or when the embedded logic is opposite, this requires relearning on the target task. And the corresponding scene of the source tasks cannot be reused through the semantic overlap. For instance, when the source task doesn't necessitate a key for door access while the target task does, The environmental description will remain consistent (`door_closed, is_env, no_key, is_agent`), but the source policy will not work. And here we will explain why $\psi(c_{\mathcal{T}}^{j}, c_{\mathcal{T}}^{j})$ is calculated, it allows the agent by fine-tuning the similarity weight to relearn the skills instead of reusing the source policies when all source policies are ineffective. Specifically, we introduce a performance function $\Phi(c_t^i, c_{\mathcal{T}}^{j})$ to record the average cumulative discounted extrinsic return $\mathcal{G}$ (Algorithm 2, lines 16–19) for the execution of the selected guidance scene c_t^i on the corresponding scene of the target task to fine-tune the similarity. Moreover, we add entropy $H(c_t^i)$ as a dynamic balance between exploration and exploitation to Φ, and ϵ is a hyper-parameter controlling the weight of $H(c_t^i)$, which is expected to focus on exploitation when the corresponding policy achieves better performance, and on the contrary, focus on exploration. The Φ is calculated as follows:

$$\begin{aligned} \Phi(c_t^i, c_{\mathcal{T}}^{j}) = \mathcal{G}(c_t^i, c_{\mathcal{T}}^{j}) + \min(0, clip(-\mathcal{G}(c_t^i), \epsilon)) * H(\pi_{c_t^i}), \\ t \in [\mathcal{S}_1, ..., \mathcal{S}_N, \mathcal{T}], i \in [1, n_t], j \in [1 : n_{\mathcal{T}}]. \end{aligned} \tag{2}$$

$\pi_{c_t^i}$ represents the probability distribution of the clauses contained in c_t^i. Combining ψ and Φ (normalized to between [0, 1]), the dual similarity measurement Λ can be calculated as follows:

$$\Lambda(c_t^i, c_{\mathcal{T}}^{j}) = \psi(c_t^i, c_{\mathcal{T}}^{j}) + \Phi(c_t^i, c_{\mathcal{T}}^{j}), t \in [\mathcal{S}_1, ..., \mathcal{S}_N, \mathcal{T}], i \in [1, n_t], j \in [1 : n_{\mathcal{T}}] \tag{3}$$

3.3 PADDLE

In this section, we propose the overall framework of PADDLE, and the specific process is shown in Algorithm.2. PADDLE refines policy reuse at the subtask level, that is, when upper level modules make decisions, they select a guiding policy from the policy library $\Pi = \{\pi_1, \ldots, \pi_N, \pi_{\mathcal{T}}\}$ based on dual similarity measurement. Specifically, in each iteration, when the high-level modules make a decision, the hybrid decision model will choose to activate a clause, and PADDLE finds the corresponding scene $c_{\mathcal{T}}^{j}$ according to the activated clause. Then, PADDLE inputs $c_{\mathcal{T}}^{j}$ into Ψ to obtain the $c_{\mathcal{S}_i}^{max}$ and $\psi(c_{\mathcal{T}}^{j}, c_{\mathcal{S}_i}^{max})$ for each task. Finally, input $c_{\mathcal{T}}^{j}$ and $c_{\mathcal{S}_i}^{max}$ into Λ to obtain the dual similarity measurement as weight, and select the policy with maximum weight as the guiding policy π_g. Note that when the measurement of the target policy is the same as that of the source policy, select the source policy. After selecting the π_g, we replace the corresponding module in the target policy with the c_g^{max} and its corresponding low-level policy to interact with the environment and use the obtained samples to train the target policy. When selecting the source task, if there are multiple similar scenes, randomly select one of them to execute. After each iteration, update the cumulative reward for π_g in each corresponding $c_{\mathcal{T}}^{j}$, and the specific process is also shown in Figure 1(c).

4 Experiments

In this section, we evaluate PADDLE on two domains, MiniGrid [10] and Maze2D [13], and construct tasks that require the agent to learn a series of skills. To better reflect real-world, we select some

Algorithm 2 PADDLE

Require: Source policies $\Pi = \{\pi_1, \ldots, \pi_n\}$, hyper-parameters γ_{head}, γ_{body} and ϵ, target policy $\pi_{\mathcal{T}}$ and performance function Φ (initialize with a constant)

Get Logic Similarity Function Ψ by Alg. 1
while episode_step $<$ max_episode_step **do**
  while not done **do**
    while select goal from high-level module **do**
      Obtain the $c^j_{\mathcal{T}}$ according to the $clause$ activated
      (id, max_similarity, set_c) $= \Psi(c^j_{\mathcal{T}})$
      for i $= 1$ to N **do**
        Get $c^{max}_{\mathcal{S}_i}$ by id=i from set_c
        W(π_i) $= \Lambda(c^{max}_{\mathcal{S}_i}, c^j_{\mathcal{T}})$
      end for
      W($\pi_{\mathcal{T}}$) $= \Lambda(c^j_{\mathcal{T}}, c^j_{\mathcal{T}})$
      $\pi_g = argmax_{\pi \in \Pi \cup \pi_{\mathcal{T}}}$W($\pi$)
      Use the c_g corresponding to π_g to select the appropriate subgoal
      Replace the low-level policy with π_g.
      Collect samples $\mathcal{S}_l = (s, a, s', \mathcal{H}, r_{intr})$ and $S_h = (B, \mathcal{H}, R_{extr})$ using π_g
      for $\mathcal{H}$ in $\mathcal{S}_h$ **do**
        Get π_g corresponding c_g, time step $t_{\mathcal{H}}$ and corresponding $c_{\mathcal{T}}$ of the target task
        $\mathcal{G}(c_g, c_{\mathcal{T}}) =$ average$(\sum_{t=t_{\mathcal{H}}}^{|S_h|} \gamma^t R^t_{extr})$
      end for
    end while
  end while
  Update $\pi_{\mathcal{T}}$ using $\mathcal{S}_l$ and $\mathcal{S}_h$
end while

simple tasks as source tasks (in Appendix) and some complex tasks as target tasks in Figure 2. On the left of Figure 2 is the MiniGrid: In the above picture, the agent is tasked with moving the yellow ball, then opening the box to retrieve the key to open the door, before finally placing the key down and picking up the green ball (BlockedBoxUnlockPickup); In the following picture, unlike BlockedBoxUnlockPickup, the agent needs to pick up the yellow ball, move it to the yellow square, and in the last step move the green ball to the red square (BlockedBoxPlaceGoal). On the right of Figure 2 is the Maze2D: The agent needs to go to [6,2] to get coffee and send it to [7,4], then go to [4,7] to get the email and send it to [7,6], and finally go to the red ball. There is some logical overlap between the source and target tasks, which is an appropriate benchmark for policy reuse.

We compare various baseline algorithms, including TL methods CUP and PTF, hierarchical algorithms [24], the original underlying algorithm, and the proposed hybrid decision-making model. This hybrid decision model is applicable to all program synthesis methods and RL algorithms, and in this work, we use GALOIS [7], PPO [32] and TD3 [14]: GALOIS-PPO and GALOIS-TD3. Moreover, the hybrid decision model can address tasks in continuous action spaces and more complex tasks. To ensure fairness, we use the same training settings for all methods. More implementation details are in the appendix.

4.1 Analysis of Transfer Performance

For better evaluation of PADDLE, we selected four transfer scenarios and two complex target tasks in the MiniGrid, and each scenario consists of two source tasks: **(A)** BlockedDoor and GapBall, **(B)** BlockedDoor and DoorBall, **(C)** BlockedDoor and BoxDoor, **(D)** BlockedDoor and OpenGoal. For BlockedBoxUnlockPickup, the source tasks in the first scenario include all the skills required, to test whether the algorithm can quickly recombine past skills. The source tasks in the second and third scenarios only include

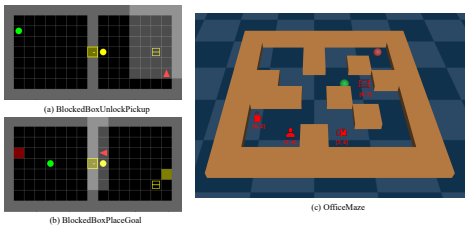


Figure 2: Experimental environments.

some of the skills required for BlockedBoxUnlockPickup, but the missing skills are in different positions in the complete skill chain, which requires the agent to recombine past skills and quickly learn new ones. The source tasks in the fourth scenario include the opposite logic of BlockedBoxUnlockPickup, requiring the agent to select useful skills and learn new ones. For BlockedBoxPlaceGoal, all four scenarios only include some of the skills required for the complete target task, but with different numbers of missing skills, to test the algorithm's ability to solve more complex problems. In the Maze2D, we considered one transfer scenario: CoffeeMaze and MailMaze, where the source tasks include all the skills required to solve the target task. We only verified the algorithm's ability in the continuous domain, as its ability to recombine past knowledge and quickly learn new knowledge has been thoroughly validated in the MiniGrid environment, and we can easily believe that PADDLE has good transfer performance in similar sequential logic tasks.

We compared PADDLE with several baseline algorithms, and all the results were averaged over six random seeds. As shown in Figure 4 and Figure 3, the baseline algorithms have difficulty achieving good learning performance in all transfer scenarios, which is due to the use of black-box policies for transfer learning. The agent cannot effectively understand which source policies may be effective in the target task, requiring additional learning costs that limit the algorithm's further performance. In contrast, the white-box knowledge with cause-effect logic can quickly enable the agent to learn which source policy may be effective at each stage, greatly reducing learning costs and improving transfer efficiency. From the experimental results, for the first transfer scenario on BlockedBoxUnlockPickup, PADDLE has good jump-start performance, mainly due to the function of the logic similarity, which indicates the ability of our method to quickly recombine past knowledge, and also proves that white-box knowledge is a better policy representation. In other transfer scenarios, PADDLE has greatly improved performance compared to the hybrid decision model learned from scratch, proving our method's ability to reuse, recombine, and re-learn knowledge.

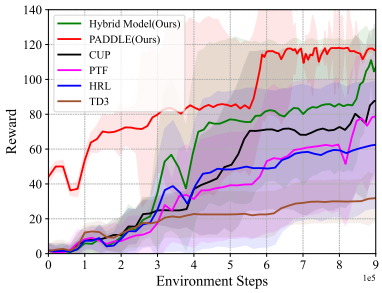


Figure 3: Results of transfer experiments in Maze2D.

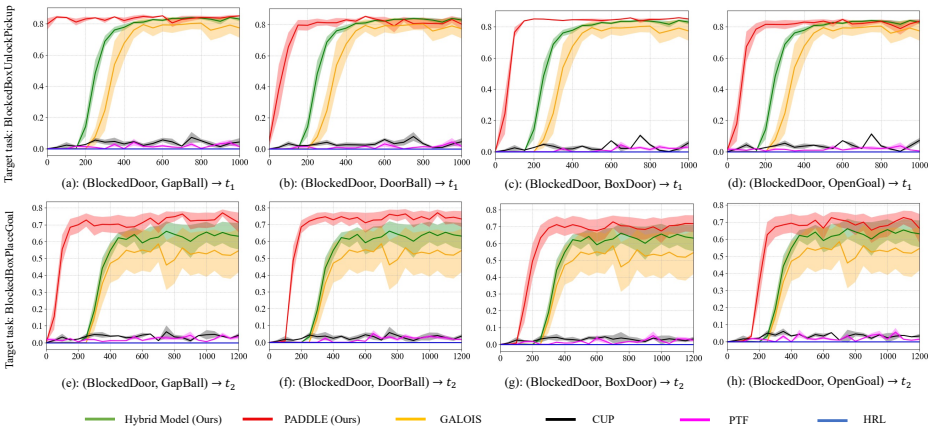


Figure 4: Results of transfer experiments in MiniGird; Within each sub-graph, the x-axis represents the number of training episodes, and the y-axis represents the normalized discounted reward converted according to the step size of the completed task.

4.2 Analysis of Reusing, Recomposing, and Re-learning Knowledge

This section provides a visualization of the source policy selection during the transfer process. The left of Figure 5 shows the weight changes of different policies selected for BlockedBoxUnlockPickup, and the two graphs above and the two graphs below are the weight changes of choosing different policies during the training process (left) and after convergence (right) under the **(A)** and the **(C)** transfer scenarios, respectively. For the upper left graph, the source policies carry more weight at

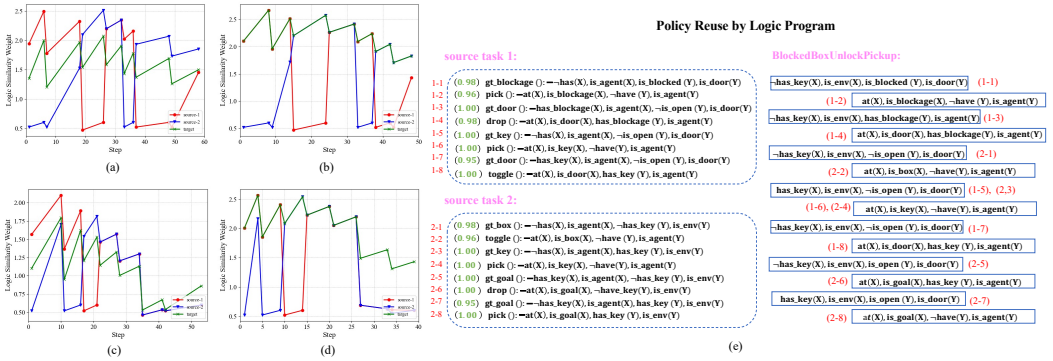


Figure 5: The interpretation of the agent's behaviors learned by PADDLE. (a)-(d) show the variations of the (unnormalized) similarity weights for each source and target policies. (e) shows a visualization of the transfer process, the green numbers is the execution probability of clause in the source task.

various stages compared to the target policy. This emphasizes that utilizing the source policy yields greater benefits, aligning well with the practical dynamics of the training process. For the upper right graph, the weight of the target policy used reaches the optimal value. This indicates that with the advancement of training, the target policy effectively acquires all the knowledge encompassed within the source task. The difference between the two graphs below is that the end of the curve is always the target policy with more weight. This is because through the dual similarity, when the skills required for the target task do not exist in the source policy, the weights will be quickly adjusted, and the target policy will be used for re-learning. To better explain the transfer process, taking the **(A)** transfer scenario as an example, we explain the process of policy switching during transfer. At the beginning of the task, the agent needs to remove the yellow ball. The body atoms returned will describe the information of related objects in the environment, and then query the high-level logic program of the source policies to see if there is the most similar fragment. The source policy (1-1) clause (describe door blockage) is the most similar and will be reused in this part. Especially, the complete visualization results are on the right of Figure 5.

4.3 Analysis of Quantity and Quality of Source Policies

Is it more difficult to discover useful knowledge during transfer as the number of source policies grows or random policies are added to the source policy pool? This further affects the difficulty of source policy library construction and limits the scenarios for transfer. Evaluate our approach on a larger set of source policies. This section evaluates the sensitivity of PADDLE to the quantity and quality of source policies, and furthermore, we also evaluate the robustness of PADDLE in handling random source policies that cannot provide meaningful candidate actions for solving the target task. Firstly, we choose three cases to test on BlockedBoxUnlockPickup. In these three cases, the number of source policies increases and the number of skills possessed also increases (1) DoorBall and OpenGoal, (2) DoorBall, OpenGoal, and BlockedDoor, (3) DoorBall, OpenGoal, BlockedDoor and BoxDoor. Meanwhile, we compare with GALOIS fine-tuning experiments [17] on the best source task (DoorBall). The results are shown in Figure 6(a). With the further improvement of the quantity and quality of the source policies, PADDLE can effectively select the appropriate source policies to guide the learning of new tasks and improve transfer efficiency. Furthermore, we add all the source policies used and the policy learned from BlockedBoxUnlockPickup to the policy library to solve BlockedBoxPlaceGoal. Note that we use all the task policies prepared on the MiniGrid environment to test whether our approach can add policies obtained from similar tasks to the policy library, so as

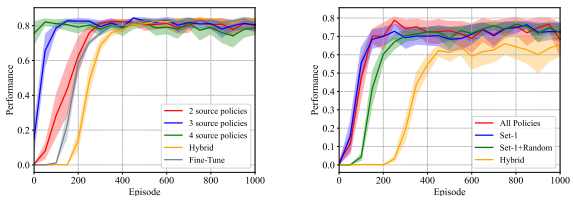


Figure 6: Analysis of quantity and quality of source policies.

to quickly solve downstream tasks and replace policies other than BlockedDoor and GapBall with random policies and add them to the source policy set, and test them in the above transfer scenario. As shown in Figure 6(b), it shows that even if there are some source policies that do not make sense, our method can still effectively select and learn appropriate guidance policies.

5 Related Work

Policy Reuse: Policy reuse [12] is a TL approach. By using the internal connection among tasks, the knowledge from previously learned policies can be used to accelerate the learning of the target task [2, 19, 9, 43]. One class of methods introduces additional components into the underlying algorithm [25, 42]. For example, PTF [42] proposes a hierarchical policy and models multi-policy transfer as the option learning problem, but training these components induces either optimization non-stationarity or heavy sampling cost, significantly impairing the effectiveness of transfer. Another main class of methods is the value-based policy or advantage-based selection method [26, 9, 37, 45]. For example, CUP [45] chooses the source policy that has the largest one-step performance improvement over the current target policy utilizing the critic, which avoids training any extra components and efficiently reuses source policies. However relying too heavily on the value function has limitations, including sensitivity to sparse or delayed rewards and value estimation errors. All previous works learn deep neural network policies/Q-tables, which are hard to reveal the logic behind. Our proposed method uses a hybrid decision model to synthesize high-level logic programs learn low-level DRL policy, and propose the logic similarity to select the appropriate source policy as the guiding policy, making it easy to detect small changes in logic levels and improving transfer efficiency.

Program Synthesis in RL: Integrating human knowledge into reinforcement learning systems is a hot topic, especially leveraging program synthesis for reinforcement learning, this type of research advocates taking a set of program specifications (e.g., natural language instructions and policy sketches, etc.) to induce an explicit program that satisfies the given specification [33, 22, 3, 20]. These methods directly synthesize a white-box logic program [44, 7] as the DRL policy and have been demonstrated to improve performance and interpretability significantly. Especially, program-guided RL breaks down complex tasks into subtasks and learns sub-policies for them. This class of methods guides agents to complete tasks with programs that explicitly specify the flow of subtasks under given environmental conditions, improving downstream generalization by expressing policies as explicit function programs. where command programs are used as a new implementation of hierarchical reinforcement learning, where the agent's policy is guided by a higher-level program [35, 46, 5, 39, 44]. Although these methods have demonstrated good generalization in simple cross-task transfer scenarios, they are limited to one-to-one transfer scenarios and target tasks with few logical changes, in addition, they only work in discrete domains. The hybrid decision model and transfer algorithm proposed in this article will effectively solve this problem.

6 Conclusion

In this paper, we propose a novel transfer framework PADDLE with a hybrid decision model as the backbone. Unlike most previous transfer methods that assumed, extracted, and used black box policy, we explore how to effectively measure the logic similarity and transfer explainable knowledge, which further improves transfer efficiency. PADDLE is simple to implement and easy to combine with existing DRL algorithms. Experimental results indicate that PADDLE outperforms previous state-of-the-art transfer methods. As for future work, it is worthwhile extending PADDLE to multiagent problems to capture the transferable knowledge among multiple agents, even heterogeneous agents. Another direction is to learn the optimal logic programs from human feedback to further release the expert knowledge assumption.

Acknowledgments and Disclosure of Funding

This work is supported by the Major Research Plan of the National Natural Science Foundation of China(Grant No. 92370132) and the National Key R&D Program of China (Grant No. 2022ZD0116402). Part of this work has taken place in the Intelligent Robot Learning (IRL) Lab at the University of Alberta, which is supported in part by research grants from the Alberta Machine

Intelligence Institute (Amii); a Canada CIFAR AI Chair, Amii; Compute Canada; Huawei; Mitacs; and NSERC.

References

[1] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[2] Mohammadamin Barekatain, Ryo Yonetani, and Masashi Hamaya. Multipolar: Multi-source policy aggregation for transfer reinforcement learning between diverse environmental dynamics. *arXiv preprint arXiv:1909.13111*, 2019.

[3] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. *Advances in neural information processing systems*, 31, 2018.

[4] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[5] Ethan Brooks, Janarthanan Rajendran, Richard L Lewis, and Satinder Singh. Reinforcement learning of implicit and explicit control flow instructions. In *International Conference on Machine Learning*, pages 1082–1091. PMLR, 2021.

[6] Tim Brys, Anna Harutyunyan, Matthew E Taylor, and Ann Nowé. Policy transfer using reward shaping. In *AAMAS*, pages 181–188, 2015.

[7] Yushi Cao, Zhiming Li, Tianpei Yang, Hao Zhang, Yan Zheng, Yi Li, Jianye Hao, and Yang Liu. GALOIS: boosting deep reinforcement learning via generalizable logic synthesis. In *NeurIPS*, pages 19930–19943, 2022.

[8] Johan Samir Obando Ceron and Pablo Samuel Castro. Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research. In *International Conference on Machine Learning*, pages 1373–1383. PMLR, 2021.

[9] Ching-An Cheng, Andrey Kolobov, and Alekh Agarwal. Policy improvement via imitation of multiple oracles. *Advances in Neural Information Processing Systems*, 33:5587–5598, 2020.

[10] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[11] Richard Evans and Edward Grefenstette. Learning explanatory rules from noisy data. *Journal of Artificial Intelligence Research*, 61:1–64, 2018.

[12] Fernando Fernández and Manuela Veloso. Probabilistic policy reuse in a reinforcement learning agent. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 720–727, 2006.

[13] Justin Fu, Aviral Kumar, Ofir Nachum, George Tucker, and Sergey Levine. D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*, 2020.

[14] Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pages 1587–1596. PMLR, 2018.

[15] Lise Getoor and Ben Taskar. *Introduction to statistical relational learning*. MIT press, 2007.

[16] Michael Gimelfarb, Scott Sanner, and Chi-Guhn Lee. Contextual policy transfer in reinforcement learning domains via deep mixtures-of-experts. In *Uncertainty in Artificial Intelligence*, pages 1787–1797. PMLR, 2021.

[17] Claire Glanois, Zhaohui Jiang, Xuening Feng, Paul Weng, Matthieu Zimmer, Dong Li, Wulong Liu, and Jianye Hao. Neuro-symbolic hierarchical rule induction. In *International Conference on Machine Learning*, pages 7583–7615. PMLR, 2022.

[18] Steven R Guberman and Patricia M Greenfield. Learning and transfer in everyday cognition. *Cognitive Development*, 6(3):233–260, 1991.

[19] Abhishek Gupta, Benjamin Eysenbach, Chelsea Finn, and Sergey Levine. Unsupervised meta-learning for reinforcement learning. *arXiv preprint arXiv:1806.04640*, 2018.

[20] Mohammadhosein Hasanbeig, Natasha Yogananda Jeppu, Alessandro Abate, Tom Melham, and Daniel Kroening. Deepsynth: Automata synthesis for automatic task segmentation in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7647–7656, 2021.

[21] León Illanes, Xi Yan, Rodrigo Toro Icarte, and Sheila A. McIlraith. Symbolic plans as high-level instructions for reinforcement learning. In *Proceedings of the 30th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 540–550. AAAI Press, 2020.

[22] Zhengyao Jiang and Shan Luo. Neural logic reinforcement learning. In *International conference on machine learning*, pages 3110–3119. PMLR, 2019.

[23] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 23(6):4909–4926, 2021.

[24] Tejas D Kulkarni, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. *Advances in neural information processing systems*, 29, 2016.

[25] Siyuan Li, Fangda Gu, Guangxiang Zhu, and Chongjie Zhang. Context-aware policy reuse. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, pages 989–997. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[26] Siyuan Li, Rui Wang, Minxue Tang, and Chongjie Zhang. Hierarchical reinforcement learning with advantage-based auxiliary rewards. *Advances in Neural Information Processing Systems*, 32, 2019.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[28] Stephen Muggleton. Inductive logic programming. *New generation computing*, 8:295–318, 1991.

[29] Athanasios S Polydoros and Lazaros Nalpantidis. Survey of model-based reinforcement learning: Applications on robotics. *Journal of Intelligent & Robotic Systems*, 86(2):153–173, 2017.

[30] Tim Rocktäschel and Sebastian Riedel. End-to-end differentiable proving. *Advances in neural information processing systems*, 30, 2017.

[31] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *arXiv preprint arXiv:1704.02532*, 2017.

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

[33] Armando Solar-Lezama. *Program synthesis by sketching*. University of California, Berkeley, 2008.

[34] Jinhua Song, Yang Gao, Hao Wang, and Bo An. Measuring the distance between finite markov decision processes. In *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems*, pages 468–476, 2016.

[35] Shao-Hua Sun, Te-Lin Wu, and Joseph J Lim. Program guided agent. In *International Conference on Learning Representations*, 2020.

[36] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[37] Yunzhe Tao, Sahika Genc, Jonathan Chung, Tao Sun, and Sunil Mallya. Repaint: Knowledge transfer in deep reinforcement learning. In *International Conference on Machine Learning*, pages 10141–10152. PMLR, 2021.

[38] Matthew E Taylor and Peter Stone. Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7), 2009.

[39] Dweep Trivedi, Jesse Zhang, Shao-Hua Sun, and Joseph J Lim. Learning to synthesize programs as interpretable and generalizable policies. *Advances in neural information processing systems*, 34:25146–25163, 2021.

[40] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

[41] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

[42] Tianpei Yang, Jianye Hao, Zhaopeng Meng, Zongzhang Zhang, Yujing Hu, Yingfeng Chen, Changjie Fan, Weixun Wang, Wulong Liu, Zhaodong Wang, and Jiajie Peng. Efficient deep reinforcement learning via adaptive policy transfer. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, pages 3094–3100. ijcai.org, 2020.

[43] Tianpei Yang, Jianye Hao, Zhaopeng Meng, Zongzhang Zhang, Yujing Hu, Yingfeng Chen, Changjie Fan, Weixun Wang, Zhaodong Wang, and Jiajie Peng. Efficient deep reinforcement learning through policy transfer. In *Proceedings of the 19th International Conference on Autonomous Agents and Multiagent Systems*, pages 2053–2055. International Foundation for Autonomous Agents and Multiagent Systems, 2020.

[44] Yichen Yang, Jeevana Priya Inala, Osbert Bastani, Yewen Pu, Armando Solar-Lezama, and Martin Rinard. Program synthesis guided reinforcement learning for partially observed environments. *Advances in neural information processing systems*, 34:29669–29683, 2021.

[45] Jin Zhang, Siyuan Li, and Chongjie Zhang. Cup: Critic-guided policy reuse. *arXiv preprint arXiv:2210.08153*, 2022.

[46] Peng Zhang, Jianye Hao, Weixun Wang, Hongyao Tang, Yi Ma, Yihai Duan, and Yan Zheng. Kogun: accelerating deep reinforcement learning via integrating human suboptimal knowledge. *arXiv preprint arXiv:2002.07418*, 2020.

[47] Matthieu Zimmer, Xuening Feng, Claire Glanois, Zhaohui Jiang, Jianyi Zhang, Paul Weng, Li Dong, Hao Jianye, and Liu Wulong. Differentiable logic machines. *arXiv preprint arXiv:2102.11529*, 2021.

A Appedix

A.1 Environments Details

MiniGrid Environments: The first five environments in Figure 7 show all the MiniGrid environments used in this paper as source tasks. Below is a detailed list of tasks for each of them. T denotes the maximum number of steps per episode and T = 320 for all MiniGrid.

- **BlockedDoor**: The agent picks up the ball in front of the door, drops it somewhere else, and gets a reward of 0.1. Then pick up the key and open the door to get the final reward.
- **GapBall**: The agent opens the box, picks up the key, and gets a reward of 0.1 and the gray gap in the middle will be accessible. then drop the key and pick up the green ball to get the final reward.
- **DoorBall**: The agent picks up the key opens the door, and gets a reward of 0.1. then drop the key and pick up the green ball to get the final reward.
- **OpenGoal**: The agent opens the door without picking up the key and reaches the green frame to get the final reward.
- **BoxDoor**: The agent opens the box, picks up the key, and gets a reward of 0.1. Then open the door to get the final reward.

In all tasks, the final reward is $r_t = 1 - 0.9(t/T)$. t denotes the time step to complete the task and this reward is given only for solving the task. The action space is discrete with seven actions: left, right, up, down, pick up, drop, and toggle. 'Toggle' unlocks a door or opens a box. The grid is procedurally generated at each episode, and the agent's initial position is random within a fixed area far from the goal. This facilitates the agent to better learn the task.

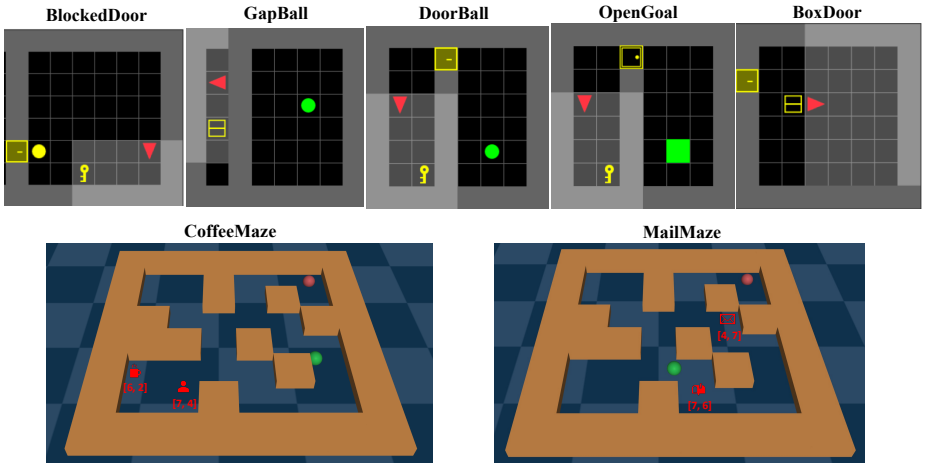


Figure 7: The first five are source task visualizations of the MiniGrid environment, and the last two are source task visualizations of the Maze2D environment

Maze2D Environments: The latter two environments in Figure 7 show all the Maze2D environments used in this paper as source tasks. Below is a detailed list of tasks for each of them and T = 1000 for all Maze2D.

- **CoffeeMaze**: The agent goes to [6,2] to get coffee and gets a reward of 10, then sends it to [7,4] and gets a reward of 10, and finally goes to the red ball to charge and get the final reward.
- **MailMaze**: The agent goes to [4,7] to get Mail and gets a reward of 10, then sends it to [7,6] and gets a reward of 10, and finally goes to the red ball to charge and get the final reward.

In all tasks, the final reward is $r_t = 100 * (1 - 0.9(t/T))$. The action space is continuous and consistent with [13]. When the agent enters a certain range of the corresponding goal, it means that

the corresponding operation is completed. The agent successfully completes the task or reaches the maximum number of steps, and the episode ends

A.2 Baseline Details

Hyper-Parameter Details: All hyper-parameters used in our experiments are listed in Table 1. We use the same set of hyper-parameters for all tasks. We use a set of hyper-parameters for all tasks in both discrete and continuous domains. Most hyper-parameters of the baselines are adjusted appropriately based on their paper.

Table 1: Detailed hyper-parameters setting for PADDLE

Hyper-Parameters	*Hyper-Parameters Values*
Discount Factor(γ_l/γ_h)	(0.99/0.95)
Optimizer	RMSProp
Learning Rate (PPO/TD3/GALOIS)	(5e-4/2e-4/0.05)
Clip Value (PPO)	0.2
Entropy Term (PPO)	0.01
Batch Size (TD3)	256
Update Frequency (TD3)	2 env steps
Policy noise (TD3)	0.2
$\gamma_{body}/\gamma_{head}$	0.2/0.8
ϵ	0.05
non-linearity	ReLU
actor/critic structure	three fully connected layers with 400 units