Learning Top-K Subtask Planning Tree based on Discriminative Representation Pre-training for Decision Making

Anonymous Author(s)
Submission Id: 215

ABSTRACT

Many complicated real-world tasks can be broken down into smaller, more manageable parts, and planning with prior knowledge extracted from these simplified pieces is crucial for humans to make accurate decisions. However, replicating this process remains a challenge for AI agents and naturally raises two questions: How to extract discriminative knowledge representation from priors? How to develop a rational plan to decompose complex problems? Most existing representation learning methods employing a single encoder structure are fragile and sensitive to complex and diverse dynamics. To address this issue, we introduce a multiple-encoder and individual-predictor regime to learn task-essential representations from sufficient data for simple subtasks. Multiple encoders are able to extract adequate task-relevant dynamics without inter-task intervention, and the shared predictor can discriminate task characteristics. We also use the attention mechanism to generate a top-K subtask planning tree, which customizes subtask execution plans in guiding complex decisions on unseen tasks. This process enables forward-looking and globality by flexibly adjusting the depth and width of the planning tree. Empirical results on a challenging platform composed of some basic simple tasks and combinatorially rich synthetic tasks consistently outperform some competitive baselines and demonstrate the benefits of our design.

KEYWORDS

Task representation, Task Planning, Reinforcement learning

ACM Reference Format:
Anonymous Author(s). 2023. Learning Top-K Subtask Planning Tree based on Discriminative Representation Pre-training for Decision Making. In *Proc. of the 22nd International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2023), London, United Kingdom, May 29 – June 2, 2023,* IFAAMAS, 17 pages.

1 INTRODUCTION

Decomposing complex problems into smaller, and more manageable subtasks with some priors and then solving them step by step is one of the critical human abilities. In contrast, reinforcement learning (RL) [2, 14, 21, 38] agents typically learn from scratch and need to collect a large amount of experience, which is expensive, especially in the real world. An efficient solution is to utilize subtask priors to extract sufficient knowledge representations and transduce them into decision conditions to assist policy learning. Many real-world challenging problems with different task structures can be solved with the subtask-conditioned reinforcement learning (ScRL) regime, such as robot skill learning [5, 10, 19, 28, 45], navigation [13, 34, 52, 53], and cooking [6, 20, 36]. ScRL endows an agent with the learned knowledge to perform corresponding subtasks, and the subtask execution path should be planned in a rational manner. Ideally, the agent should master some basic competencies to finish some subtasks, and composite the existing subtasks to find an optimal decision path to solve previously unseen and complicated problems. For example, if the final goal is to perform an unseen task like cutting onions, the agent should be able to prepare a chopping board, pick up an onion, and use a knife. For a clear understanding, we visualize two different task structures by recombining subtask sequences, as shown in Figure 1.

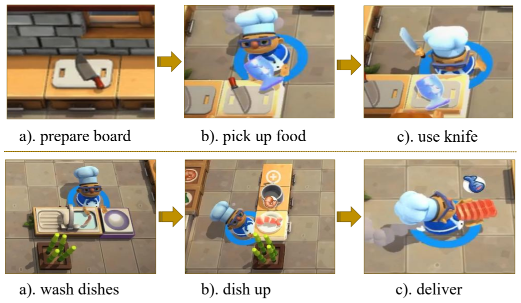


Figure 1: Two scenarios in Overcooked AI [6]: the top denotes cutting food, and the bottom is serving customers.

There are two challenges in ScRL. A fundamental question is how to construct proper knowledge representations about subtasks to transfer to more complicated tasks. The content and structure of knowledge representation are vital to the reasoning of the decision-making, since they refer to functional descriptions of different subtasks to reach the final goal. Another is how to utilize the learned knowledge to assist policy training. Most previous works [16, 30, 35, 48] mainly focus on zero-shot generalization on unseen tasks of the same scale, instead of recombining prior knowledge to plan more complicated tasks for guiding policy learning.

To address the above issues, our key idea is to construct proper knowledge representations, recombine existing knowledge, and apply it in policy training to improve training efficiency and model performance. We first introduce a learning regime based on multiple-encoder and individual-predictor to extract distinguishable subtask representations with task-essential properties. This module aims to exploit underlying task-specific features and construct proper knowledge representations using contrastive learning and model-based dynamic prediction. Compared to encoding all of the subtasks with only one encoder, the regime that each encoder corresponds to each sub-task can alleviate the burden of inter-task intervention to some extent. Next, we propose to generate a top-K subtask planning tree to customize the subtask execution plan for the agent. Then we use the subtask-conditioned reinforcement learning framework to obtain the decision policy. The top-K subtask planning tree can grow to any width and depth for solving complicated tasks of different scales. Long-term planning for the future can prevent the agent from moving towards local optima as far as possible.

The contributions of this paper can be summarized as follows.

- We introduce a multiple-encoder and individual-predictor learning regime to learn distinguishable subtask representations with task-essential properties.
- We propose to construct a top-K subtask planning tree to customize subtask execution plans for learning with subtask-conditioned reinforcement learning.
- Empirical evaluations on several challenging navigation tasks show that our method can achieve superior performance and obtain meaningful results consistent with intuitive expectations.

2 RELATED WORK

Task representation. One line of research, including HiP-BMDP [49], MATE [37], MFQI [9], CARE [40], CARL [4], etc, is designed to achieve fast adaptation by aggregating experience into a latent representation on which policy is conditioned to solve the zero/few-shot generalization in multi-task RL. Another line, such as CORRO [48], MerPO [26], FOCAL [24], FOCAL++ [23], etc, mainly focuses on learning the task/context embeddings with fully offline meta-RL to mitigate the distribution mismatch of behavior policies in training and testing. Among all the aforementioned approaches, this paper is the most related to the latter. Part of our work is devoted to finding the proper subtask representations using offline datasets to assist online policy learning.

Task planning. Task planning is essentially breaking down the final goals into a sequence of subtasks. Based on the style of planning, we classify the existing approaches into three categories: planning with priors, planning on latent space, and planning with implicit knowledge. First, planning with priors will generate one or more task execution sequences according to the current situation and the subtask function prior, such as [11, 22, 25, 31], but is limited by subtasks divided by human labor. Second, planning on latent space [1, 17, 33, 51] endows agents with the ability to reason over an imagined long-horizon problem, but it is fundamentally bottlenecked by the accuracy of the learned dynamics model and the horizon of a task. Third, some works like [18, 46, 47], utilize the modular combination for implicit knowledge injection to reach indirect task guidance and planning, which lack explainability. Our work aims to obtain proper knowledge representation extracted from the subtask itself, and generate a rational and explainable subtask tree including one or more task execution sequences.

Subtask-conditioned RL (ScRL). ScRL requires the agent to make decisions according to different subtasks. A line of work [7, 12, 27, 29, 41, 42, 50] focuses on leveraging the prefined or learned abstract subtasks, which lack transparency and are difficult to solve long-horizon problems. There is some work that tends to develop some explainable method. From the perspective of neuro symbol, BPNS [43] learns neuro-symbolic skills and use heuristic search A* to guide the decision-making. [39] learns user-interpretable capacities from manually collected trajectories and user's vocabulary. However, how to exploit these subtasks to make long-term planning is not considered, which is very unfavorable for complex tasks. However, the existing works mainly solve the short-horizon subtask and rarely focus on the recombination of the learned subtask to solve the complicated long-horizon missions. We propose an explicit subtask representation and scheduling framework to provide long-horizon planning and guidance.

3 METHODOLOGY

In this section, we first describe our problem setup. Then we introduce how to learn distinguishable subtask representations as the cornerstone. Next, we propose to construct a top-K subtask planning tree to customize subtask execution plans for the agent. After these, we introduce subtask-guided decision-making.

3.1 Problem Setup

Temporal Subtask-conditioned Markov Decision Process (TSc-MDP). We consider augmenting a finite-horizon Markov decision process [15] $(\mathcal{S}, \mathcal{A}, p, r, \rho_0, \gamma)$ with an extra tuple $(\{\mathcal{T}^i\}_{i=0}^n, \rho_s)$ as the temporal subtask-conditioned Markov decision process, where $\mathcal{S}$, $\mathcal{A}$ denote the state space and the action space, respectively. $\{\mathcal{T}^i\}_{i=0}^n$ is the subtask set and n is the number of subtasks. $p(s_{t+1}, \mathcal{T}_{t+1}^i | s_t, \mathcal{T}_t^i, a_t)$ is the dynamics function that gives the distribution of the next state s_{t+1} and next subtask $\mathcal{T}_{t+1}^i$ at state s_t to execute the subtask $\mathcal{T}_t^i$ and take action a_t. $r : \mathcal{S} \times \mathcal{T} \times \mathcal{A} \rightarrow \mathbb{R}$ denotes the reward function. ρ_0 and ρ_s denote the initial state distribution and the subtask distribution. $\gamma \in (0, 1)$ is the discount factor. Formally, the optimization objective is to find the optimal policy π and the subtask planner sp to maximize the expected cumulative return:

$$J(\pi, sp) = \mathbb{E}_{\substack{a_t \sim \pi(\cdot|s_t) \\ \mathcal{T}_t^i \sim \mathrm{sp}(\cdot|\tau)}} \left[\sum_t \gamma^t r(s_t, \mathcal{T}_t^i, a_t) \right], \quad (1)$$

where τ is the history transitions, and sp is the temporal subtask planner that generates subtask sequences at a coarse time scale.

3.2 Distinguishable Subtask Representations

Data Collection. The data collection statistics are shown in Table 3, of which 1/3 is collected from the converged PPO policy, 1/3 is collected from the intermediate trained checkpoint of PPO, and the last 1/3 is from the random policy. The intuition for such collection is to eliminate the effect of the behavior policies and collect diverse and relevant data. The collected data format for subtask $\mathcal{T}^i$ can be denoted as $\{(s_t, a_t, r_t, s_{t+1})_{t=1}^L\}^i$, where L is the trajectory length.

Subtask	#Total Episodes	#Total Transitions
PickupLoc	3000	210375
PutNext	3000	559201
Goto	3000	218191
Open	3000	149349

Table 1: The data collection statistics

Representation Pre-training. In the reinforcement learning domain, the environmental system is characterized by a transition function and a reward function. Therefore, what we need to do is to use the collected trajectories to mine the essential properties of subtasks and make proper representations from the perspective of the transition and the reward function. To learn accurate and discriminative subtask representations, we introduce a multiple-encoder and individual-predictor learning regime to extract task-essential properties, as shown in Figure 2.

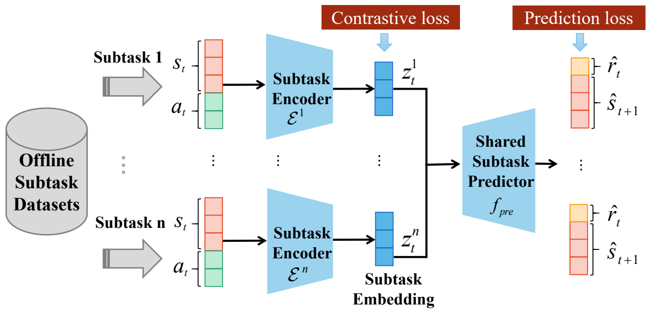

Contrastive loss
Prediction loss
Subtask 1
Offline Subtask Datasets
Subtask n
Subtask Encoder $\mathcal{E}^1$
Subtask Encoder $\mathcal{E}^n$
Shared Subtask Predictor f_{pre}
Subtask Embedding

Figure 2: The multiple-encoder and individual-predictor learning regime. Each subtask encoder $\mathcal{E}^i$ is used to encode the state-action pair $(s_t, a_t)^i \in \mathcal{T}^i$ and output the compact subtask representation z_t^i. The shared subtask predictor is used to predict the reward r_t and the next state s_{t+1}. The contrastive and prediction losses are designed to learn the many-to-one framework.

There are two design principles: i) Each subtask has its own different responsibilities and skills, so multiple encoders can boost the diversity among them. On the other hand, multiple subtask encoders, which each corresponds to one subtask, can effectively avoid mapping two state-action pairs from different subtasks with various reward and transition distributions to similar abstractions. ii) Figuring out the essential properties of transition and reward function is the key to obtaining accurate and generalizable subtask representations, so we design an auxiliary prediction task to simulate the reward and transition distributions. The shared subtask predictor is used to predict the reward and the next state so as to implicitly learn the world model for a better understanding of the accurate essential properties of the subtasks.

Supposed that we have n subtasks, we initialize n subtask encoders parameterized by $\{\theta^i\}_{i=1}^n$, respectively. First, we sample a minibatch of b transitions $\{\tau_j^i = (s_j, a_j, r_j, s_{j+1})^i\}_{j=1}^b$ from some subtask $\mathcal{T}^i$, and feed the state-action pairs $\{(s_j, a_j)^i\}_{j=1}^b$ to the subtask encoder $\mathcal{E}^i : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{Z}$. Then the latent variables $z^i \in \mathcal{Z}$ for subtask $\mathcal{T}^i$ are obtained, denoted as $z^i = \mathcal{E}^i((s,a)^i)$.

For better distinguishment, contrastive loss is applied. Similar to [32, 48], we give a definition of anchors, positive, and negative samples. Using the latent variable $q = \mathcal{E}^i(s,a)$ from the subtask $\mathcal{T}^i$ as an anchor, the $k^+ = \mathcal{E}^i(s^+, a^+)$ and $k^- = \mathcal{E}^j(s^-, a^-)$ respectively constitute positive and negative instances, where (s^+, a^+) and (s^-, a^-) are sampled from the subtask $\mathcal{T}^i$ and other subtasks $\mathcal{T}^j, j \neq i$. Therefore, the contrastive learning objective [32] for the multiple subtask encoders is:

$$\mathcal{L}_c^i = -\log \frac{\exp(q \cdot k^+)}{\exp(q \cdot k^+) + \sum_{k^-} \exp(q \cdot k^-)}. \tag{2}$$

Then adopting mini-batch sampling for training, we can optimize the contrastive objective to extract the essential variance of transition functions across different tasks and mine the shared and compact features of the same subtasks.

As for the shared subtask predictor f_{pre}, we use the mean squared error (MSE) loss to calculate the mean squared differences between true and prediction as follows.

$$\mathcal{L}_p = \lambda_r ||\hat{r} - r||_2^2 + \lambda_s ||\hat{s}' - s'||_2^2 \,, \tag{3}$$

where $\hat{r}, \hat{s}' = f_{pre}(z)$ is the prediction of the immediate reward r and the next state s'. λ_r and λ_s are positive scalars that tune the relative weights between the loss terms.

With contrastive learning for pushing inter-tasks and pulling intra-task and dynamics prediction for mining nature properties, the total loss is summarized as follows.

$$\min \sum_{subtask\ i} \sum_{\substack{\tau^i \sim \mathcal{T}^i \\ \tau^{\{n\}\setminus i} \sim \mathcal{T}^{\{n\}\setminus i}}} (\mathcal{L}_c^i + \mathcal{L}_p), \tag{4}$$

where $\tau = (s, a, r, s')$, and $\{n\}$ are the collected transitions and the total subtask set.

After finishing the first step toward compact representations for subtasks, the subtask tree can be generated.

3.3 Top-K Subtask Planning Tree Generation

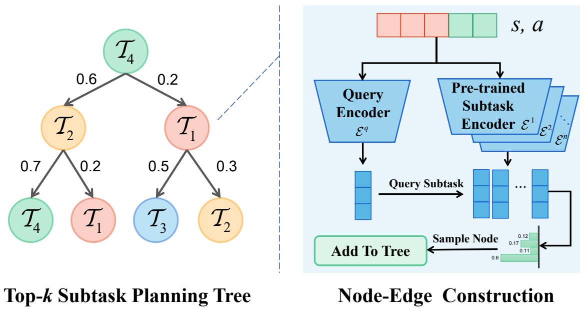


Figure 3: The left part is a top-2 subtask tree[1]which is generated by the process of the right part.

In real human society, effective decision-making always relies on good judgment of the current state, and future prediction and imagination. Enlightened by this ability of humanity, we should generate a top-K subtask tree to assist the policy training. Due to the long horizon and sparse feedback of complicated tasks, we dynamically adjust the subtask tree generation during the episode instead of using a specified tree. specifically, we will regenerate a new subtask tree while the subtask plan terminated. As shown in figure 3, we generate a relatively short-horizon subtask tree with some inference of dynamics in a coarse-to-fine manner and a new subtask tree will regenerate once the current tree has been executed to a leaf node. Next, we describe the generation process in detail.

First, we provide the detailed construction procedure of the subtask tree in a top-down manner. Here, we have a query encoder $\mathcal{E}^q$ and the n pre-trained encoder $\{\mathcal{E}^i\}_{i=1}^n$ corresponding to each subtask defined in section 3.2.

Node-edge construction process. We use the last state-action pair (s_{t-1}, a_{t-1}) to compute the query vector v^q as the anchor to retrieve the most similar subtask.

$$v^q = \mathcal{E}^q(s_{t-1}, a_{t-1}). \tag{5}$$

Next, the subtask embeddings from the pre-trained encoder can be obtained as follows.

$$\{v^i\}_{i=1}^n = \{\mathcal{E}^i(s_{t-1}, a_{t-1})\}_{i=1}^n. \tag{6}$$

[1]Note: The probability sum is not equal to 1 because we sample 2 subtasks from the original distribution and ignore other subtasks.

We utilize the attention mechanism [3] to compute the correlation score between the query vector v^q and the pre-trained subtask embeddings $\{v^i\}_{i=1}^n$. Then the normalized coefficient $\alpha_{q,i}$ is defined:

$$\alpha_{q,i} = \frac{\exp(\beta_{q,i})}{\sum_j \exp(\beta_{q,j})}, \; where \; \beta_{q,i} = v^q W_\phi^T W_\varphi v^i, \tag{7}$$

where W_ϕ, W_φ are learnable parameters and $\beta_{q,i}$ is the attention score. Here, we can model the correlation between query vector v^q and an arbitrary number of subtasks, which allows us to easily increase the number of subtasks in our tree generation process. Then the categorical distribution can be modeled using the weights from which the variable C can be sampled:

$$C \sim \mathcal{P}_\beta = Categorical\left(\frac{\exp(\beta_{q,i})}{\sum_{j \in \mathcal{N}} \exp(\beta_{q,j})}, i \in \mathcal{N} = \{1, ..., n\}\right). \tag{8}$$

Then, given the hyperparameter $K < n$, we can sample the top-K subtask ids without replacement from the categorical distribution which means sampling the first element, then renormalizing the remaining probabilities to sample the next element, etcetera. Let $C_1^*, ..., C_K^*$ be an (ordered) sample without replacement from the Categorical $\mathcal{P}_\beta$, then the joint probability can be formulated as follows.

$$P(C_1^* = c_1^*, ..., C_K^* = c_K^*) = \prod_{i=1}^{K} \frac{\exp(\beta_{q,c_i^*})}{\sum_{\iota \in \mathcal{N}_i^*} \exp(\beta_{q,\iota})}, \tag{9}$$

where $c_1^*, ..., c_K^* = argtop\text{-}K(\mathcal{P}_\beta)$[1] is the sampled top-$K$ realization of variables $C_1^*, ..., C_K^*$, and $\mathcal{N}_i^* = \mathcal{N} \backslash \{c_1^*, ..., c_{i-1}^*\}$ is the domain (without replacement) for the i-th sampled element.

Then, we use the sampled top-K subtask ids as the k nodes at a certain layer in the subtask tree (Please note that we set $k = 1$ while generating the root node). Moreover, given the hyperparameter N, we can obtain the tree of depth N by repeating the above process based on the planning transition (described in the next paragraph), and the original correlation weight $\alpha_{q,i}$ is set as connections (edges) between layer and layer. For a clear understanding, we summarize the overall construction procedure shown in figure 4.

Planning for future: m-step predictor. While expanding the subtask tree, we should make planning for the m-step prediction of the future. We design a m-step predictor denoted as $p_m(\hat{s}_{t+m}|\hat{s}_t, \hat{a}_t)$ that is used to derive the next m^{th} step's state. The action $\hat{a}_{t+m}$ is output by the policy network π (defined in Equation (15)) using the current subtask representation. The N-times rollout starting with s_{t-1} can be written as:

$$\{(s_{t-1}, a_{t-1}), (\hat{s}_{t+m-1}, \hat{a}_{t+m-1}), ..., (\hat{s}_{t+N\cdot m-1}, \hat{a}_{t+N\cdot m-1})\}.$$

Then we can use this imagined rollout to expand the subtask tree to a greater depth. That is, as each layer of depth increases, the planning will perform m-step imagination forward.

Optimization for tree generation. The optimization includes two parts: an m-step predictor p_m and an attention module. The former p_m can be trained by minimizing the mean squared error function $\mathcal{L}_m$ defined as follows. Sampled some state-action pairs

[1] $argtop\text{-}K(\mathcal{P}_\beta)$ means the corresponding subtask ids of top k probabilities from $\mathcal{P}_\beta$.

```
BUILD_TREE( $\tau$, $\mathcal{E}^q$, $\{\mathcal{E}^i\}_{i=1}^n$, $K$, $N$)
1. Initilize $K$, $N$, $Tree=None$; // tree width K, tree depth N
2. $Tree$ = ADD_TREE( $Tree$, 0, 1, $\tau$, $\mathcal{E}^q$, $\{\mathcal{E}^i\}_{i=1}^n$ )
3. for $l=1$ to $N$ do
4.     $Tree$ = ADD_TREE( $Tree$, $l$, $K$, $\tau$, $\mathcal{E}^q$, $\{\mathcal{E}^i\}_{i=1}^n$ )
5. return $Tree$
```

```
ADD_TREE( $Tree$, $l$, $K$, $\tau$, $\mathcal{E}^q$, $\{\mathcal{E}^i\}_{i=1}^n$ )
1. for $N_i$ in $Node_l$: // Traverse the $Node_l$ ( nodes in layer l )
2.     $\tau_{new}$ = $planning(\tau, l, N_i)$ // plan l times
3.     $P_{node}$ = Attention( $\tau_{new}$, $\mathcal{E}^q$, $\{\mathcal{E}^i\}_{i=1}^n$ ) // Node picking distribution
4.     Picking Top-K from $P_{node}$;
5.     Add corresponding subtask id to the tree node;
6.     Connect edges with corresponding probability;
7. return $Tree$
```

Figure 4: The basic top-K subtask tree construction process.

from replay buffer $\mathcal{D}$, we have:

$$\mathcal{L}_m = \mathbb{E}_{\substack{(s_t, a_t, s_{t+m}) \sim \mathcal{D} \\ \hat{s}_{t+m} = p_m(s_t, a_t)}} \left[||\hat{s}_{t+m}, s_{t+m}||_2^2\right]. \tag{10}$$

The attention module formulated by Equation (7), (8), (9), and parameterized by $\omega = \{\phi, \varphi\}$ can be optimized in a Monte-Carlo policy gradient manner [44]. First, we define the intra-subtask cumulative reward $R_t = \sum_{j=0}^{\tilde{T}-1} \gamma^j r_{t+j}$ during the interval $\tilde{T}$ of the subtask execution process as follows. By decomposing the cumulative rewards into a Bellman Expectation equation in a recursive manner, we acquire:

$$G(s_t, \mathcal{T}_t) = \mathbb{E}\left[R_{t+1} + \gamma G(s_{t+\tilde{T}}, \mathcal{T}_{t+\tilde{T}})|s_t, \mathcal{T}_t\right], \tag{11}$$

where s_t and $\mathcal{T}_t$ is the current state and the executed subtask, γ is the discounted factor, and $\mathcal{T}_{t+\tilde{T}}$ is the next subtask.

Then, by applying the mini-batch technique to the off-policy training, the gradient updating can be approximately estimated as:

$$\omega \leftarrow \omega + \alpha \nabla \log \mathcal{P}_\beta(\mathcal{T}_t|s_{t-1}, a_{t-1}) \cdot G(s_t, \mathcal{T}_t), \tag{12}$$

where t denotes the timestep when a subtask execution begins and α is the learning rate.

3.4 Tree-auxiliary Policy

Given a generated top-K subtask tree, we can obtain a tree-auxiliary policy as follows. Generally, agents should have the foresight to judge the current situation, give a rational plan, and then use the plan to guide policy learning. Therefore, the first step is to generate a plan based on the top-K subtask tree.

Subtask execution plan generation. Here, we design a discounted upper confidence bound (d-UCB) method for the path finding. For each subtask $\mathcal{T}^i$, we record its selection count $c_{\mathcal{T}^i}$ and the average cumulative rewards $r_{\mathcal{T}^i}$ while the subtask terminates. The UCB value for the $\mathcal{T}^i$ in the planning tree is computed as:

$$v_{\mathcal{T}^i} = r_{\mathcal{T}^i} + 0.5 \cdot \sqrt{\frac{2 \ln c_{\mathcal{T}^i}^{pa}}{c_{\mathcal{T}^i}}}, \tag{13}$$

where $c^{pa}_{\mathcal{T}^i}$ is the selection count of the parent node of $\mathcal{T}_i$.

Then, we give a definition of the discounted path length via the evaluation of d-UCB. We adopt the greedy policy to pick the path (subtask execution plan) with the maximum discounted length. The examples for designing d-UCB is shown in Appendix A.2.

Definition 1. *Sample a path from root to leaf in the tree, the length of the discounted path can be defined as follows.*

$$\hat{d}(root \rightarrow leaf) = \sum_l \kappa^l e^{l,l+1} \cdot v_{\mathcal{T}^i},$$

where $e^{l,l+1}$ is a sampled edge between l^{th} layer and $(l+1)^{th}$ layer, the value of the edge is the probability of subtask transition, $\kappa = 0.8$ is the discounted factor, and $v_{\mathcal{T}^i}$ is the UCB value of the sampled node in the $(l+1)^{th}$ layer.

Subtask termination condition. Subtask execution should be terminated when the current situation changes, and we give a measurement about when to terminate the current subtask. we first calculate the cosine similarity of the current query vector $\mathcal{E}^q(s_t, a_t)$ and the corresponding subtask embedding $\mathcal{E}^{\mathcal{T}_{id}}(s_t, a_t)$ and the similarity function is defined as follows.

$$Sim(\mathcal{E}^q(s_t, a_t), \mathcal{E}^{\mathcal{T}_{id}}(s_t, a_t)) = \frac{\left((\mathcal{E}^q(s_t, a_t))^T \mathcal{E}^{\mathcal{T}_{id}}(s_t, a_t)\right)}{(||\mathcal{E}^q(s_t, a_t)|| \cdot ||\mathcal{E}^{\mathcal{T}_{id}}(s_t, a_t)||)}. \tag{14}$$

For every timestep, the agent will compare the similarity score of the current timestep and the last timestep. A score of 0 or 1 means the direction of two vectors are orthogonal or identical, respectively. Avoiding the drastic changes of subtasks, we choose $\Delta = 0.5$ as the division. If the similarity score is less than 0.5, the subtask will terminate and turn to the next one until the path ends.

Subtask-guided decision making. In our implementation, the policy will consider the current state and the subtask embedding and give a comprehensive decision, formulated as follows.

$$a_t \sim \pi(\cdot|s_t, \mathcal{E}^{\mathcal{T}_{id}}(s_{t-1}, a_{t-1})). \tag{15}$$

As for the policy optimization, we adopt the PPO-style [38] training regime, and we refer readers to Appendix A.9 for a detailed definition. We summarized the objective function as follows.

$$\mathcal{L}_\pi = [\min(i_t \cdot A_t, clip(i_t, 1-\epsilon, 1+\epsilon)A_t)], \tag{16}$$

where $i_t = \frac{\pi(a_t|s_t, \mathcal{E}^{\mathcal{T}_{id}}(s_{t-1}, a_{t-1}))}{\pi_{old}(a_t|s_t, \mathcal{E}^{\mathcal{T}_{id}}(s_{t-1}, a_{t-1}))}$ is the importance ratio between the current policy π and last policy π_{old}, A_t is is an estimator of the advantage function at timestep t, $\epsilon = 0.2$ is the clipping parameter for PPO, and the function clip() is used to limit i_t in an interval $[1-\epsilon, 1+\epsilon]$.

All in all, the overall optimization flow including subtask representation pre-training, top-K subtask generation, and tree-auxiliary policy is summarized in Algorithm 1.

4 EXPERIMENTS

In this section, we seek to empirically evaluate our method to verify the effectiveness of our solution. We design the experiments to 1) verify the effectiveness of the multiple-encoder and individual-predictor learning regime; 2) test the performance of our top-K subtask tree on the most complicated scenarios in the 2D navigation task BabyAI [8]; 3) conduct some sensitivity analyses in our settings.

Algorithm 1: Overall Procedure

```
Ensure subtask representation policy $\rho = \{\{\mathcal{E}^i\}_{i=1}^n, f_{pre}\}$,
  tree building policy $\mathcal{E}^q$, and tree-axuliary policy $\pi$;
// Subtask Representation Pre-training
Collect diverse subtask datasets $\{\mathcal{T}^1, ..., \mathcal{T}^n\}$;
for each iteration do
    Sample a subtask $\mathcal{T}^i$ and transitions $\{\tau_t^i\}_{t=1}^b$, $\{\tau_t^{i,+}\}_{t=1}^b$;
    $\{z_t^i\}_{t=1}^l = \{\mathcal{E}^i(s_t^i, a_t^i)\}_{t=1}^l$ ;
    $\{z_t^{i,+}\}_{t=1}^l = \{\mathcal{E}^i(s_t^{i,+}, a_t^{i,+})\}_{t=1}^l$ ;
    Sample other subtasks $\mathcal{T} \backslash \mathcal{T}^i$ and transitions $\{\tau_t^{i,-}\}_{t=1}^b$;
    $\{z_t^{i,-}\}_{t=1}^l = \{\mathcal{E}^-(s_t^{i,-}, a_t^{i,-})\}_{t=1}^l$ ;
    Predict reward $r$ and next state $s'$;
    Update $\rho$ to minimize Eq. (4);
end
// Tree Generation and Tree-auxiliary Policy
for each episode do
    Initial state-action pair $s\_a_{cur} = (s_0, a_0)$;
    BUILD_TREE($s\_a_{cur}, \mathcal{E}^q, \{\mathcal{E}^i\}_{i=1}^n$) ; // Fig. 4
    Use d-UCB in Sec. 3.4 to find a plan $\zeta$;
    for each timestep $t$ do
        if $\zeta$ is None then
            Regenerate the tree and the plan;
        end
        while $\zeta$ is not None do
            $\mathcal{T}_{id} = \zeta[0]$;
            // execute current subtask Select action
              $a_t = \pi(\cdot|s_t, \mathcal{E}^{\mathcal{T}_{id}}(s\_a_{cur}))$;
            Receive reward $r$ and next state $s'$ ;
            if $Sim(\mathcal{E}^q(s_t, a_t), \mathcal{E}^{\mathcal{T}_{id}}(s_t, a_t)) < 0.5$ then
                remove $\mathcal{T}_{id}$ from $\zeta$; // execute next subtask
            end
        end
    end
    Update $\pi$ with Eq. (15);
    Update $\mathcal{E}^q$ with Eq. (10) and Eq. (12);
end
```

4.1 Experimental Setting

We choose BabyAI as our experimental platform, which incorporates some basic subtasks and some composite complex tasks. For each scenario, there are different small subtasks, such as opening doors of different colors. The agent should focus on the skill of opening doors, rather than the color itself. Moreover, the BabyAI platform provides by default a $7 \times 7 \times 3$ symbolic observation o_t (a partial and local egocentric view of the state of the environment) and a variable length instruction c as inputs at each time step t.

Basic Subtask. BabyAI contains four basic subtasks that may appear in more complicated scenarios, including *Open*, *Goto*, *PutNext*, and *Pickup*, as shown in figure 5. (a) ***Open*** : Open the door with a certain color and a certain position, e.g. "open the green door on your left". The color set and position set are defined as : $COLOR = \{red, green, blue, purple, yellow, grey\}$ and $POS = \{on\ your\ left, on\ your\ right, in\ front\ of\ you, behind\ you\}$. (b) ***Goto*** : Go to an object that the subtask instructs, e.g. "go to red ball"

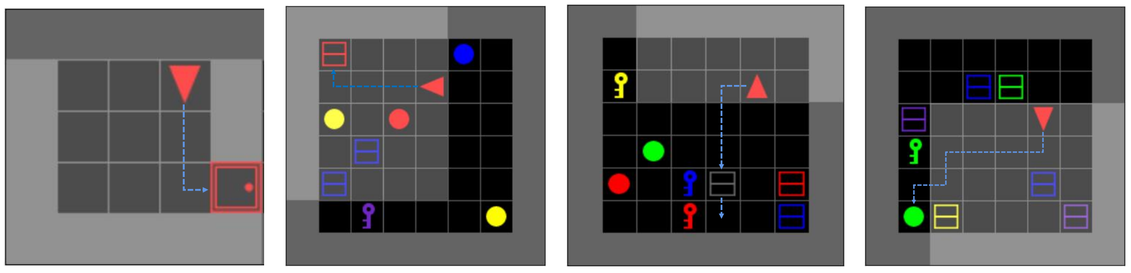

(a) Open (b) Go to (c) Put next (d) Pick up

Figure 5: The schematics of four basic subtasks. The dashed lines denote the path to reach the goal. The agent represented by the red triangle is partially observable to the grid and the light-grey shaded area represents its field of view. The missions are described as : (a). Open the red door; (b). Go to the red box; (c). Put the grey box next to the red key. (d). Pick up the green ball.

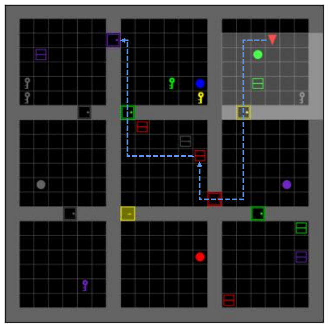


Figure 6: The illustration for one of the BossLevel scenario. The mission is: "pick up a red box and go to the purple door".

means going to any of the red balls in the maze. The object set is defined as: $OBJ = \{door,\ box,\ ball,\ key\}$. (c) ***PutNext*** : Put objects next to other objects, e.g. "put the blue key next to the green ball". (d) ***PickupLoc*** : Pick up some objects, e.g."pick up a box".

Complicated Task. The BabyAI platform includes some tasks that require three or more subtasks compositions to solve, which are called complicated tasks. What we want to do is to solve these tasks through our tree-based policy to reach higher performance and faster convergence. Take the "BossLevel" scenario for example, the agent should follow one of the instructions like "pick up the grey box behind you, then go to the grey key and open a door", which contains three subtasks and is shown as figure 6. More detailed descriptions are shown in Appendix A.3.

Baselines. In this paper, we select the state-of-the-art (SOTA) algorithm PPO [38] and SAC [14], the tree-based planning algorithm SGT [17], and the context-based RL method CORRO [48] as baselines to verify the effectiveness of our proposed method. PPO is a popular deep reinforcement learning algorithm for solving discrete and continuous control problems, which is an on-policy algorithm. SAC is an off-policy actor-critic deep RL algorithm based on the maximum entropy reinforcement learning framework. SGT adopts the divide-and-conquer style that recursively predicts intermediate sub-goals between the start state and goal. It can be interpreted as a binary tree that is most similar to our settings. CORRO is based on the framework of context-based meta-RL, which adopts a transition encoder to extract the latent context as a policy condition.

Parameter Settings. Additionally, we provide the description of all the neural networks used in our framework and more detailed experimental hyperparameter settings in Appendix A.4 and Appendix A.5, respectively.

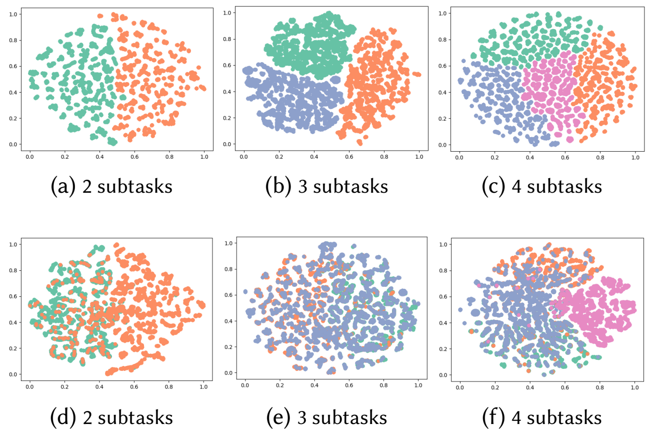
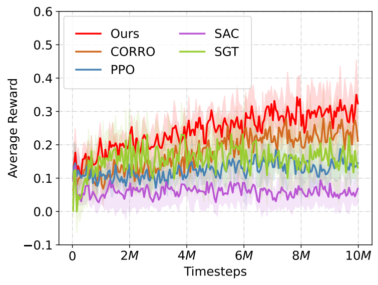

(a) 2 subtasks (b) 3 subtasks (c) 4 subtasks
(d) 2 subtasks (e) 3 subtasks (f) 4 subtasks

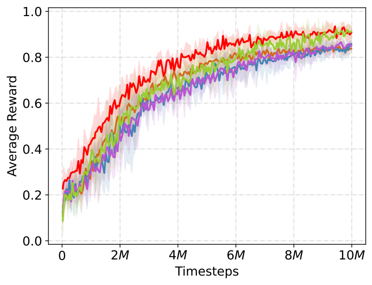
Figure 7: The t-SNE visualization of subtask embeddings. (a-c): The results are generated by our multiple encoders and one predictor learning regime on 2, 3, and 4 subtasks, respectively. (d-f): The visualization for the shared encoder and one decoder learning framework.

4.2 Results on Subtask Representaion

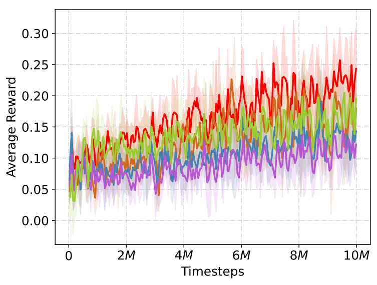
Here, we report the visualization results on the settings described in Section 3.2 to verify the effectiveness of our distinguishable subtask representations, shown in Figure 7. To verify the effectiveness of our design, the multiple-encoder module is replaced with a shared encoder. For example, for 3 subtasks, three encoders respectively encode 3 subtasks in our setting. As a comparison, we use one shared encoder to encode all the subtasks. After training our model and the ablated model on 2, 3, and 4 subtasks, respectively, Figure 7 give the t-SNE results of subtask embeddings for these six models.



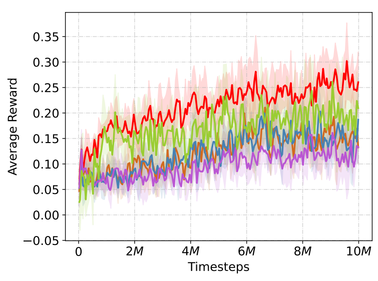
(a) GoToSeq



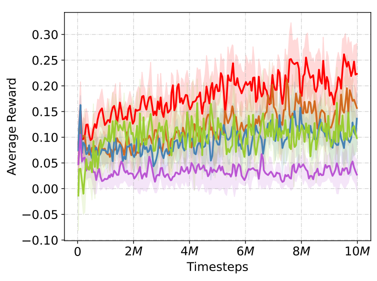
(b) GoToSeqS5R2



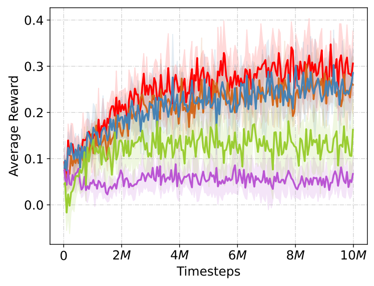
(c) SynthSeq



(d) BossLevel



(e) BossLevelNoUnlock



(f) MiniBossLevel

Figure 8: The training curve comparisons on six complicated tasks, averaging over 10 independent running seeds.

The results in Figure 7(a-c) show that our regime is capable of clearly illustrating separate clusters of embeddings when we respectively train our method on 2, 3, and 4 subtasks with 2, 3, and 4 encoders. However, Figure 7(d-f) indicate that only one shared encoder can not learn the boundaries and distinguishment clearly. We speculate mixed representation training for multiple subtasks would interfere with the stabilization of the representation module.

Moreover, due to the powerful capacity of contrastive learning and model-based dynamic prediction, our model can exhibit strong representational discrimination and contain enough dynamics knowledge, which will be verified empirically as follows.

4.3 Results on Training Curves

To evaluate our method in complicated and dynamic environments, we conduct various experiments on multiple difficulty levels of BabyAI, elaborated in Appendix A.3. These levels require multiple competencies to solve them. For example, *GotoSeq* only requires navigating a room or maze, recognizing the location, and understanding the sequence subtasks while *BossLevel* requires mastering all competencies including navigating, opening, picking up, putting next, and so on.

Here, we report the average episode reward by training steps for each level, averaging over 10 independent running seeds to plot the reward curve with standard deviation. In *GoToSeqS5R2* which is the relatively simplest level in the six, all the test rewards keep increasing as the training steps increase and almost reach the best finally. In other complicated tasks, our algorithm consistently obtains higher rewards than all the baselines in the different levels of BabyAI shown in Figure 8, while the popular SOTA like PPO and SAC almost fail. This indicates that our method is capable of macro-planning and robust enough to handle complicated tasks by decomposing the overall process into a sequence of execution subtasks.

Although we only trained 10M timesteps due to the limitation of computing resources, our method has a higher starting point than the baseline model, and it has been rising. However, the gain of the baselines at the same timestep is not obvious in most cases. This significant improvement is mainly due to two aspects. On the one hand, the adequate knowledge representation facilitates the inference capacity of our model and gives a high-level starting point for warming up the training. On the other hand, the learned subtask planning strategy can make rational decisions while confronting various complicated situations.

4.4 Results on Execution Performance

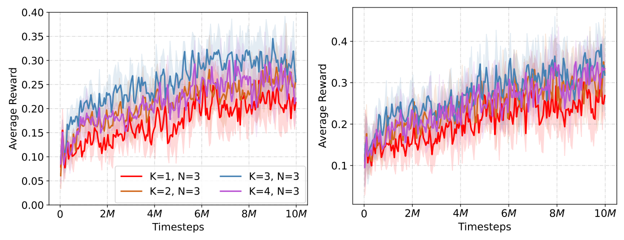
To further test the execution performance of all methods, we freeze the trained policies and execute for 100 episodes, and report the mean and standard deviations in Table 2. Our algorithm can consistently and stably outperform all the competitive baselines by a large margin, which demonstrates the benefits of our framework.

4.5 Results on Various Tree Width

After the verification of the training and execution performance of our technique, we perform in-depth sensitivity analyses regarding the width of the planning tree shown in Figure 9. Here, we freeze the tree depth as 3, same as the main experiments. Additional experimental results about other scenarios can be found in Appendix A.6.

Table 2: Execution performance on various complicated tasks. We report the average reward with standard deviations of 100 testing episodes with best values in bold.

	Ours	CORRO	PPO	SAC	SGT
GoToSeq	**0.30**±0.07	0.23± 0.06	0.14± 0.04	0.05± 0.04	0.15± 0.08
GoToSeqS5R2	**0.91**± 0.02	0.84± 0.02	0.84± 0.03	0.85± 0.02	**0.91**± 0.04
SynthSeq	**0.22**± 0.05	0.16± 0.05	0.12± 0.05	0.12± 0.04	0.18± 0.05
BossLevel	**0.26**± 0.05	0.15± 0.04	0.14± 0.05	0.12± 0.04	0.19± 0.06
BossLevelNoUnlock	**0.24**± 0.04	0.17± 0.04	0.10± 0.05	0.04± 0.03	0.11± 0.05
MiniBossLevel	**0.29**± 0.06	0.26± 0.05	0.25± 0.05	0.06± 0.03	0.14± 0.06



(a) BossLevel (b) GotoSeq

Figure 9: The sensitivity analyses on various tree width.

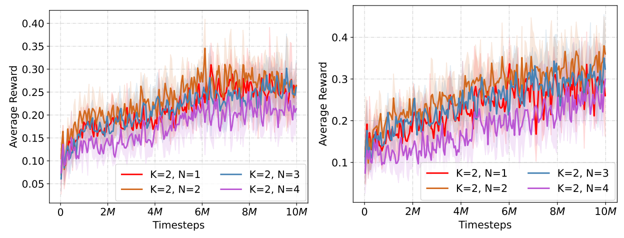
The horizontal axis represents the training steps, and the vertical axis is the average episode reward with a standard deviation, averaging across 10 independent running seeds. According to our intuition, the wider the tree is, the more diverse the subtask distribution is. However, according to the experimental results, we find that the gain is not the largest when the width of the tree is the largest. We speculate that as the width of the tree is larger, more information can be selected, but the pertinence of problem-solving is not strong, which is easy to cause confusion. Therefore, it is critical to choose an appropriate tree width and it is our future direction to conduct the adaptive selection.

4.6 Results on Various Tree Depth

As shown in Figure 10, we further provide in-depth sensitivity analyses to understand the behavior of our planning tree depending on various tree depths. Here, we freeze the tree width as 2, the same as the main experiments. Then we provide more sensitivity analyses in Appendix A.7. The depth represents the capacity for predicting the future of our planning tree. As shown in Figure 10, the results show that when the depth $N = 2$ or $N = 3$, our model wins the comparable performance also. However, the curves about $N = 4$ show a degraded performance than others. We analyze that the prediction error about the future at every step keeps accumulating, so it may make the model go in a bad direction although the foresight ability can help the model make a global decision.

4.7 Results on Subtask Execution Statistics

As shown in Figure 11, we visualize the subtask proportion on six complicated tasks after testing 1000 episodes respectively. It is surprisingly consistent with the pre-defined task missions. It basically corresponds to predefined tasks, which intuitively justifies the rationality of our approach. For example, *GoToSeq* and *GoToSeqS5R2* only require the competency of executing some sequences about"Go To", our model focuses on this subtask during the execution in most cases. There exists some error that other tasks will be selected occasionally, and we speculate that it is because of the inaccuracy of forward prediction during expanding the depth of the tree. Our model executes subtasks at roughly the same frequency when dealing with the other four tasks which require comprehensive competencies.



(a) BossLevel (b) GotoSeq

Figure 10: The sensitivity analyses on various tree depth.

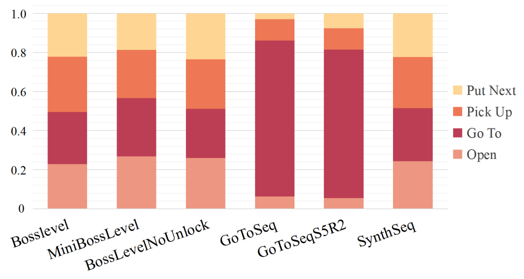


Figure 11: The bars represent the mean proportion of each subtask executed by our trained model over 1000 episodes.

5 CONCLUSION

In this paper, we have considered the difficulties of task planning to solve complicated problems, and propose a two-stage planning framework. The multiple-encoder and individual-predictor regime can extract discriminative subtask representations with dynamic knowledge from priors. The top-k subtask planning tree, which customizes subtask execution plans to guide decisions, is important in complicated tasks. Generalizing our model to unseen and complicated tasks demonstrates significant performance and exhibits greater potential to deploy it to real-life complicated tasks. More sensitivity analyses also demonstrate the capacity of our planning tree.

For future works, it would be interesting to automatically learn more unseen and explainable subtasks besides the prior subtasks, and make influential and global planning as a reference to human decisions in real-life applications.

REFERENCES

[1] Shuang Ao, Tianyi Zhou, Guodong Long, Qinghua Lu, Liming Zhu, and Jing Jiang. 2021. CO-PILOT: COllaborative Planning and reInforcement Learning On sub-Task curriculum. *Advances in Neural Information Processing Systems* (2021).

[2] Mohammad Babaeizadeh, Iuri Frosio, Stephen Tyree, Jason Clemons, and Jan Kautz. 2016. Reinforcement learning through asynchronous advantage actor-critic on a gpu. *arXiv preprint arXiv:1611.06256* (2016).

[3] Dzmitry Bahdanau, Kyung Hyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. *International Conference on Learning Representations* (2015).

[4] Carolin Benjamins, Theresa Eimer, Frederik Schubert, André Biedenkapp, Bodo Rosenhahn, Frank Hutter, and Marius Lindauer. 2021. Carl: A benchmark for contextual and adaptive reinforcement learning. *arXiv preprint arXiv:2110.02102* (2021).

[5] Stephane Cambon, Rachid Alami, and Fabien Gravot. 2009. A hybrid approach to intricate motion, manipulation and task planning. *The International Journal of Robotics Research* 28, 1 (2009), 104–126.

[6] Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. 2019. On the utility of learning about humans for human-ai coordination. *Advances in neural information processing systems* (2019).

[7] Elliot Chane-Sane, Cordelia Schmid, and Ivan Laptev. 2021. Goal-conditioned reinforcement learning with imagined subgoals. *International Conference on Machine Learning* (2021).

[8] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. 2019. BabyAI: First Steps Towards Grounded Language Learning With a Human In the Loop. *International Conference on Learning Representations* (2019).

[9] Carlo D'Eramo, Davide Tateo, Andrea Bonarini, Marcello Restelli, Jan Peters, et al. 2020. Sharing knowledge in multi-task deep reinforcement learning. *International Conference on Learning Representations* (2020).

[10] Xiaofeng Gao, Ran Gong, Tianmin Shu, Xu Xie, Shu Wang, and Song-Chun Zhu. 2019. VRKitchen: An interactive 3D environment for learning real life cooking tasks. (2019).

[11] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. 2021. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems* (2021).

[12] Dibya Ghosh, Abhishek Gupta, and Sergey Levine. 2018. Learning actionable representations with goal-conditioned policies. *arXiv preprint arXiv:1811.07819* (2018).

[13] Himanshu Gupta, Bradley Hayes, and Zachary Sunberg. 2022. Intention-Aware Navigation in Crowds with Extended-Space POMDP Planning. *International Conference on Autonomous Agents and Multiagent Systems* (2022).

[14] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. 2018. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905* (2018).

[15] Ronald A Howard. 1960. Dynamic programming and markov processes. (1960).

[16] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. 2022. Bc-z: Zero-shot task generalization with robotic imitation learning. *Conference on Robot Learning* (2022), 991–1002.

[17] Tom Jurgenson, Or Avner, Edward Groshev, and Aviv Tamar. 2020. Sub-Goal Trees a Framework for Goal-Based Reinforcement Learning. *International Conference on Machine Learning* (2020).

[18] Leslie Pack Kaelbling and Tomás Lozano-Pérez. 2017. Learning composable models of parameterized skills. *International Conference on Robotics and Automation* (2017).

[19] Kei Kase, Chris Paxton, Hammad Mazhar, Tetsuya Ogata, and Dieter Fox. 2020. Transferable task execution from pixels through deep planning domain learning. *International Conference on Robotics and Automation* (2020).

[20] Paul Knott, Micah Carroll, Sam Devlin, Kamil Ciosek, Katja Hofmann, Anca D Dragan, and Rohin Shah. 2021. Evaluating the robustness of collaborative agents. *arXiv preprint arXiv:2101.05507* (2021).

[21] Vijay Konda and John Tsitsiklis. 1999. Actor-critic algorithms. *Advances in neural information processing systems* (1999).

[22] James J Kuffner and Steven M LaValle. 2000. RRT-connect: An efficient approach to single-query path planning. *International Conference on Robotics and Automation* (2000).

[23] Lanqing Li, Yuanhao Huang, Mingzhe Chen, Siteng Luo, Dijun Luo, and Junzhou Huang. 2021. Provably Improved Context-Based Offline Meta-RL with Attention and Contrastive Learning. *arXiv preprint arXiv:2102.10774* (2021).

[24] Lanqing Li, Rui Yang, and Dijun Luo. 2020. Focal: Efficient fully-offline meta-reinforcement learning via distance metric learning and behavior regularization. *arXiv preprint arXiv:2010.01112* (2020).

[25] Jacky Liang, Mohit Sharma, Alex LaGrassa, Shivam Vats, Saumya Saxena, and Oliver Kroemer. 2022. Search-based task planning with learned skill effect models for lifelong robotic manipulation. *International Conference on Robotics and Automation* (2022).

[26] Sen Lin, Jialin Wan, Tengyu Xu, Yingbin Liang, and Junshan Zhang. 2021. Model-Based Offline Meta-Reinforcement Learning with Regularization. *International Conference on Learning Representations* (2021).

[27] Neville Mehta, Prasad Tadepalli, and A Fern. 2005. Multi-agent shared hierarchy reinforcement learning. *ICML Workshop on Rich Representations for Reinforcement Learning* (2005), 45.

[28] Katharina Mülling, Jens Kober, Oliver Kroemer, and Jan Peters. 2013. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research* 32, 3 (2013), 263–279.

[29] Soroush Nasiriany, Vitchyr Pong, Steven Lin, and Sergey Levine. 2019. Planning with goal-conditioned policies. *Advances in Neural Information Processing Systems* (2019).

[30] Aviv Netanyahu, Tianmin Shu, Joshua Tenenbaum, and Pulkit Agrawal. 2022. Discovering Generalizable Spatial Goal Representations via Graph-based Active Reward Learning. *International Conference on Machine Learning* (2022).

[31] Takato Okudo and Seiji Yamada. 2021. Online Learning of Shaping Reward with Subgoal Knowledge. In *International Conference on Autonomous Agents and MultiAgent Systems*.

[32] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748* (2018).

[33] Karl Pertsch, Oleh Rybkin, Frederik Ebert, Shenghao Zhou, and et al. 2020. Long-horizon visual planning with goal-conditioned hierarchical predictors. *Neural Information Processing Systems* (2020).

[34] Hao Quan, Yansheng Li, and Yi Zhang. 2020. A novel mobile robot navigation method based on deep reinforcement learning. *International Journal of Advanced Robotic Systems* 17, 3 (2020), 1729881420921672.

[35] Heechang Ryu, Hayong Shin, and Jinkyoo Park. 2022. REMAX: Relational Representation for Multi-Agent Exploration. *International Conference on Autonomous Agents and Multiagent Systems* (2022).

[36] Bidipta Sarkar, Aditi Talati, Andy Shih, and Dorsa Sadigh. 2022. PantheonRL: A MARL Library for Dynamic Training Interactions. *AAAI Conference on Artificial Intelligence* (2022).

[37] Lukas Schäfer, Filippos Christianos, Amos Storkey, and Stefano V Albrecht. 2022. Learning task embeddings for teamwork adaptation in multi-agent reinforcement learning. *arXiv preprint arXiv:2207.02249* (2022).

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

[39] Tom Silver, Ashay Athalye, Joshua B Tenenbaum, Tomás Lozano-Pérez, and Leslie Pack Kaelbling. 2022. Learning Neuro-Symbolic Skills for Bilevel Planning. In *6th Annual Conference on Robot Learning*.

[40] Shagun Sodhani, Amy Zhang, and Joelle Pineau. 2021. Multi-task reinforcement learning with context-based representations. *International Conference on Machine Learning* (2021).

[41] Sungryull Sohn, Junhyuk Oh, and Honglak Lee. 2018. Hierarchical reinforcement learning for zero-shot generalization with subtask dependencies. *Advances in Neural Information Processing Systems* (2018).

[42] Richard S Sutton, Marlos C Machado, G Zacharias Holland, David Szepesvari Finbarr Timbers, Brian Tanner, and Adam White. 2022. Reward-Respecting Subtasks for Model-Based Reinforcement Learning. *arXiv preprint arXiv:2202.03466* (2022).

[43] Pulkit Verma, Shashank Rao Marpally, and Siddharth Srivastava. 2022. Discovering User-Interpretable Capabilities of Black-Box Planning Agents. In *AAAI 2022 Workshop on Explainable Agency in Artificial Intelligence*.

[44] Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning* 8, 3 (1992), 229–256.

[45] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. 2008. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine* 29, 1 (2008), 9–9.

[46] Mingyu Yang, Jian Zhao, Xunhan Hu, Wengang Zhou, and Houqiang Li. 2022. LDSA: Learning Dynamic Subtask Assignment in Cooperative Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2205.02561* (2022).

[47] Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. 2020. Multi-task reinforcement learning with soft modularization. *Advances in Neural Information Processing Systems* (2020).

[48] Haoqi Yuan and Zongqing Lu. 2022. Robust Task Representations for Offline Meta-Reinforcement Learning via Contrastive Learning. *International Conference on Machine Learning* (2022).

[49] Amy Zhang, Shagun Sodhani, Khimya Khetarpal, and Joelle Pineau. 2020. Learning Robust State Abstractions for Hidden-Parameter Block MDPs. *International Conference on Learning Representations* (2020).

[50] Han Zhang, Jingkai Chen, Jiaoyang Li, Brian C Williams, and Sven Koenig. 2022. Multi-Agent Path Finding for Precedence-Constrained Goal Sequences. *International Conference on Autonomous Agents and Multiagent Systems* (2022).

[51] Lunjun Zhang, Ge Yang, and Bradly C Stadie. 2021. World model as a graph: Learning latent landmarks for planning. *International Conference on Machine Learning* (2021).

[52] Kai Zhu and Tao Zhang. 2021. Deep reinforcement learning based mobile robot navigation: A review. *Tsinghua Science and Technology* 26, 5 (2021), 674–691.

[53] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. 2017. Target-driven visual navigation in indoor scenes using deep reinforcement learning. *International Conference on Robotics and Automation* (2017).

A APPENDIX

A.1 Detailed Statistics about the Collected Data

The data collection statistics are shown as Table 3, of which 1/3 is collected from the converged PPO policy, 1/3 is collected from the intermediate trained checkpoint of PPO, and the last 1/3 is from the random policy.

Subtask	#Total Episodes	#Total Transitions
PickupLoc	3000	210375
PutNext	3000	559201
Goto	3000	218191
Open	3000	149349

Table 3: The data collection statistics

A.2 Detailed Description about the Intuition for d-UCB

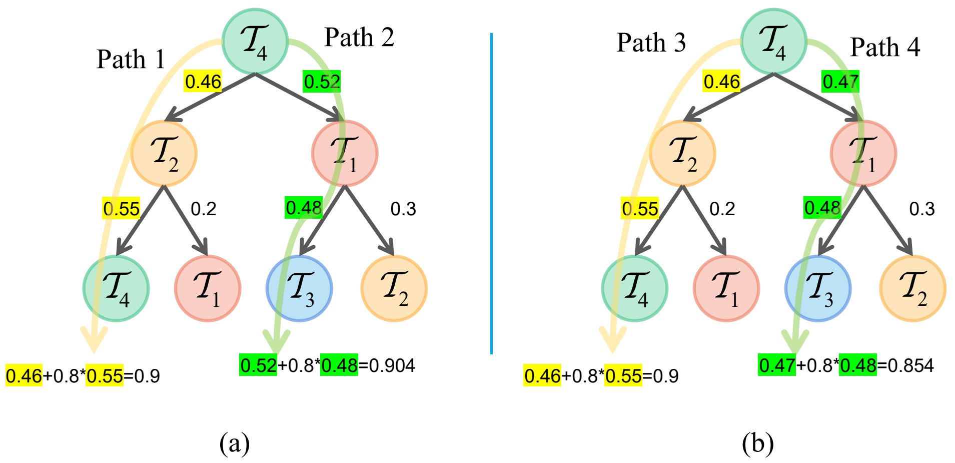


Figure 12: Two instance trees are visualized to verify the effect of d-UCB described in Section 3.4. To ignore the effect of UCB value, we set the value of all nodes as 1.

Here, we provide another two methods. One is called max selection (MS), which means that MS will pick the path in that the cumulative probability is maximum. Next, we will compare it with our d-UCB method. Another is called greedy selection (GS), which means that GS will pick the current maximum probability layer by layer. Next, we will compare it with our d-UCB method.

As shown in Figure 12(a), MS will select path 1, which obviously places a greater weight on the future with greater uncertainty. However, our d-UCB will choose path 2, which tends to maximize the immediate benefit by discounting the future uncertain score, which is consistent with our intuition.

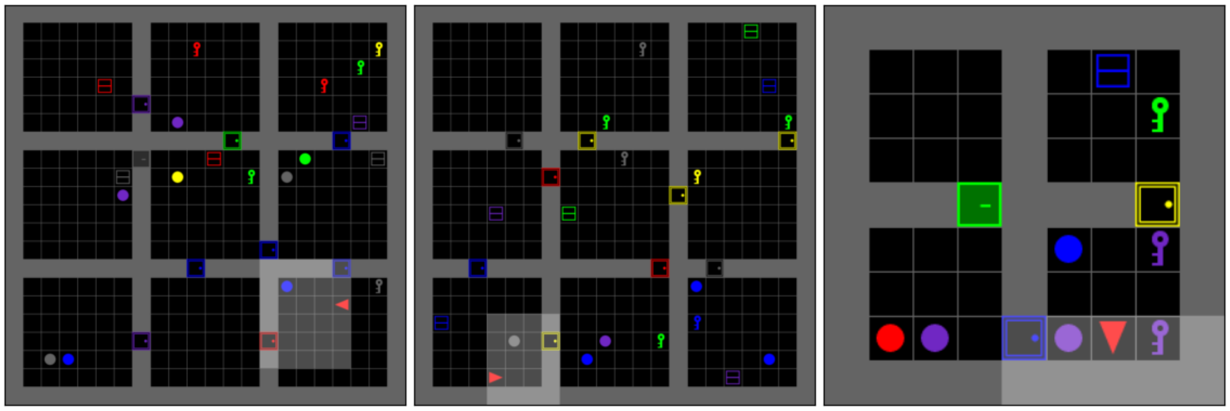
Moreover, as shown in Fig. 12(b), not like MS tending to pick the edge with the highest probability, our d-UCB will choose path 3. Our d-UCB also considers the optimal subtask to perform in the future when the nearest paths exhibit similar transition probabilities.

A.3 Detailed Description about Complicated Tasks

BossLevel. The level requires the agent to implement any union of instructions produced according to Baby Language grammar, it involves the content of all other levels. For example, "open a door and pick up the green key after you go to a ball and put a red box next to the red ball" is a command of the level. This level requires all the competencies including navigating, unblocking, unlocking, opening, going to, picking up, putting next, and so on.

BossLevelNoUnlock. The doors of all rooms are unlocked which the agent can go through without unlocking by a corresponding key. For example, the command "open a purple door, then go to the yellow box and put a yellow key next to a box" doesn't need the agent to get a key before opening a door. This level requires all the competencies like BossLevel except unlocking.

MiniBossLevel. The level is a mini-environment of BossLevel with fewer rooms and a smaller size. This level requires all the competencies like BossLevel except unlocking.


(a) ‘BabyAI-BossLevel-v0’ (b) ‘BabyAI-BossLevelNoUnlock-v0’ (c) ‘BabyAI-MiniBossLevel-v0’

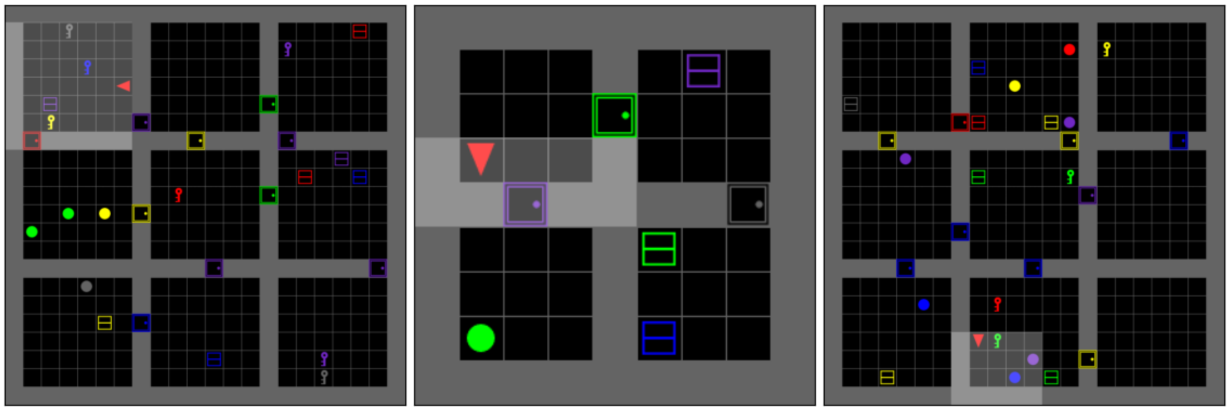

(d) ‘BabyAI-GoToSeq-v0’ (e) ‘BabyAI-GoToSeqS5R2-v0’ (f) ‘BabyAI-SynthSeq-v0’

Figure 13: The diagrams for six complicated tasks in BabyAI.

GoToSeq. The agent needs to complete a series of go-to-object commands in sequence. For example, the command "go to a grey key and go to a purple key, then go to the green key" only includes go-to instructions. GotoSeq only requires navigating a room or maze, recognizing the location, and understanding the sequence subtasks.

GoToSeqS5R2. The missions are just like GoToSeq, which requires the only competency of “go to”. Here, S5 and R2 refer to the size and the rows of room, respectively.

SynthSeq. The commands, which combine different kinds of instructions, must be executed in order. For example, "open the purple door and pick up the blue ball, then put the grey box next to a ball". This level requires the competencies like GotoSeq except inferring whether to unlock or not.

A.4 Network Architecture

There is a summary of all the neural networks used in our framework about the network structure, layers, and activation functions.

Please note that ’-’ denote that we refer readers to check the open source repository about FiLMedBlock and CausalSelfAttention, see https://github.com/caffeinism/FiLM-pytorch and https://github.com/sachiel321/Efficient-Spatio-Temporal-Transformer, respectively.

A.5 Parameter Settings

There are our hyper-parameter settings for the training of distinguishable subtask representation and the tree-based policy, shown in Table 5 and Table 6, respectively.

	Network Structure	Layers	Hidden Size	Activation Functions
Subtask Encoder	MLP	6	[256, 256, 128, 128, 32]	ReLu
Shared Subtask Predictor	MLP	3	[64, 128]	ReLu
Query Encoder	MLP	4	[32, 16, 8]	ReLu
Tree Node Generation	CausalSelfAttention	1	-	Softmax
Feature Extractor	CNN+FiLMedBlock	-	-	ReLu
Policy Actor	MLP	2	64	Tanh
Policy Critic	MLP	2	64	Tanh

Table 4: The Summary for Network Architecture

Description	Value
lr	$3 * 10^{-4}$
contrastive batch size	64
negative pairs number	16
task embedding size	5
λ_r	0.9
λ_s	1.0
encoder hidden size	[256, 256, 128, 128, 32]
decoder hidden size	[64, 128]

Table 5: Task Representation Hyper-paramters

Description	Value
optimizer	Adam
α	10^{-4}
β_1	0.9
β_2	0.999
ε-greedy ε	10^{-5}
clipping ϵ	0.2
seed	$[0, 10)$
number of process	64
total frames	$10M$
lr	10^{-4}
batch size	1280
eval interval	1
eval episodes	500
image embedding	128
memory of LSTM	128
query hidden size	[32, 16, 8]
number of head	1
discounted factor of UCB	0.9
m-step predictor	5

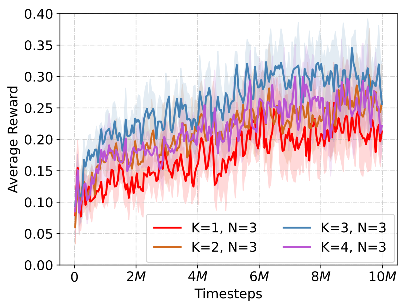
Table 6: Tree-based Policy Hyper-paramters

A.6 Additional Experimental Results on Tree Width

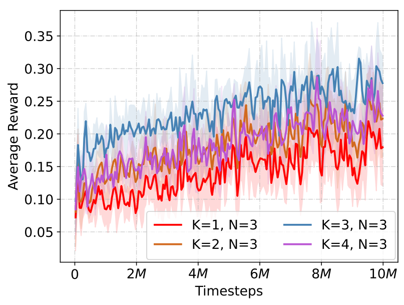
As shown in Figure 14, the other four experiments basically presented results consistent with the analysis in the main text. Among them, the experimental environment of *GoToSeqSR2* is relatively simpler, and the performance of our model basically reaches the optimal trend, so the ablation study of tree width does not seem to work. We believe that our model is capable of handling complex tasks and is not very discriminative for simple tasks. In addition, for *MiniBossLevel*, the experiment with a tree width of 2 achieves the best performance. It indicates that our model can already show a good performance improvement in a slightly simpler environment with a tree width of 2.

A.7 Additional Experimental Results on Tree Depth

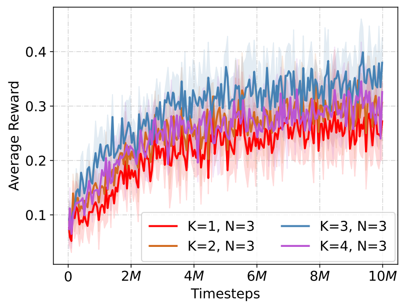
As shown in Figure 15, we plot the training curves of the other four complicated tasks freezing the width as 2. Combining the experiments in the main text, these results all show a common phenomenon: the larger the value of N, the worse the performance of our model. This cumulative effect of errors shows that the accuracy of our tree-building prediction model needs to be improved, and it is also a defect of our model. We will focus on how to improve the forward prediction ability of the model in future work. In addition, changing the depth of the tree in the scene *GoToSeqS5R2* still cannot reflect the difference in the model. This shows that our model can achieve excellent performance even if our tree prediction module is not very accurate. On the one hand, this success may depend on the strong fault tolerance of our pre-trained representation model, and the generated representations can correctly guide model learning; on the other hand, it may depend on the design of our subtask termination function, which can stop the subtask in time when the task mismatch is found.



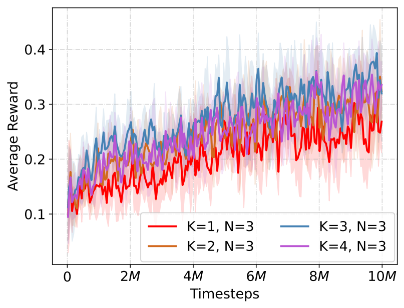
(a) BossLevel



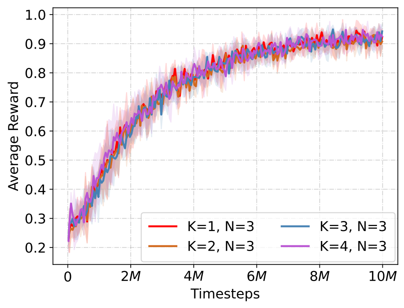
(b) BossLevelNoUnlock



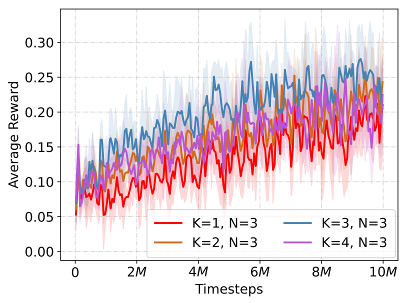
(c) MiniBossLevel



(d) GoToSeq

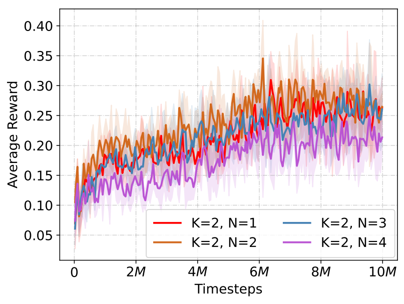


(e) GoToSeqS5R2

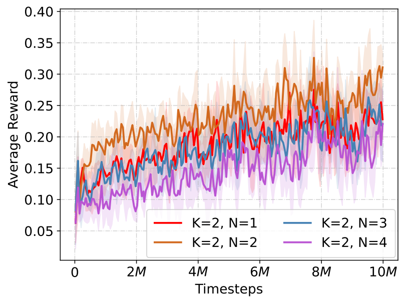


(f) SynthSeq

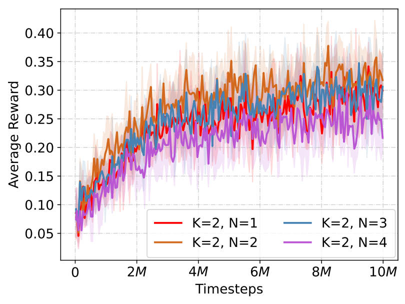
Figure 14: The sensitivity analysis of tree width for other four complicated tasks in BabyAI.



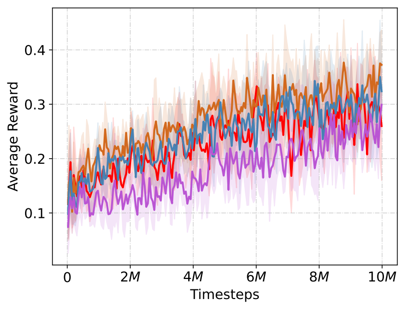
(a) BossLevel



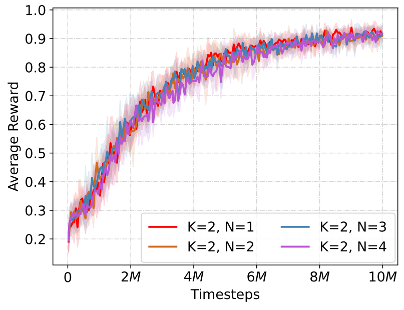
(b) BossLevelNoUnlock



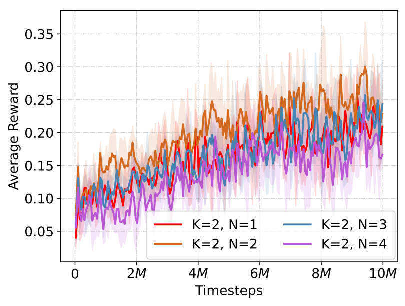
(c) MiniBossLevel



(d) GoToSeq



(e) GoToSeqS5R2



(f) SynthSeq

Figure 15: The sensitivity analysis of tree depth for other four complicated tasks in BabyAI.

A.8 Additional Experimental Results on the Threshold of Similarity Score

We have conducted a set of experiments about the threshold as follows.

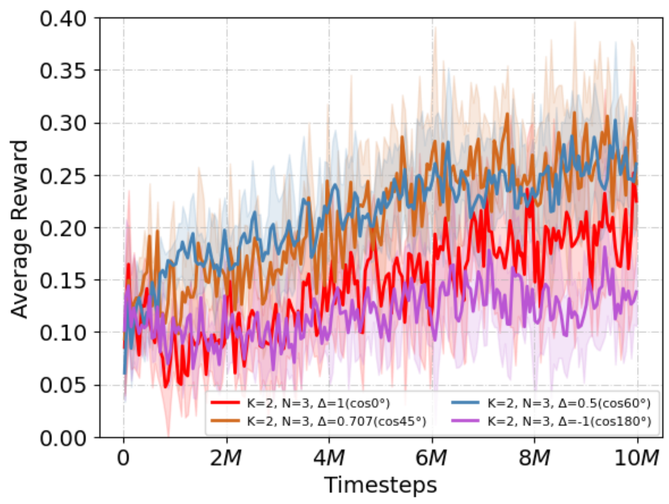


Figure 16: The sensitivity analysis about the threshold of the similarity score on BossLevel.

The explanation of the cosine similarity score:

- Cosine similarity scores range from -1.0 to 1.0, which corresponds to 180° to 0° of the angle between two vectors.
- The **larger** the **similarity score**, the **smaller** the **angle** between the two vectors, that is, the higher the cosine similarity, the slower the subtask changes.
- Some special angles: cos0° = 1, two vectors have the same direction. cos90° = 0, two vectors are orthogonal. cos180° = -1, two vectors have opposite directions.

The experimental results show that:

- The purple line: Δ = cos180° = -1, means that never change the subtask in one episode. It shows the worst performance, which indicates that there should have a correct subtask to guide policy learning.
- The red line: Δ = cos0° = 1, means that always change the subtask in one episode and shows the bad performance. Changing the subtask every time is not a good choice because it will increase the uncertainty of policy learning, moreover, it brings a larger time cost and training burden.
- The blue line: Δ = cos60° = 0.5, and the brown line: Δ = cos45° = 0.707, has the similar performance. We choose the blue line (Δ = cos60° = 0.5) as our default configuration because it can decrease the time cost by controlling the change of subtasks not too drastically.
- Furthermore, automatically picking a proper threshold configuration to balance the training complexity and performance will be an interesting research challenge.

A.9 Some Preliminaries

A.9.1 Proximal policy optimization (PPO) . Proximal Policy Optimisation (PPO) [1] is a recent advancement in the field of Reinforcement Learning, which shows remarkable performance when it was implemented by OpenAI.

PPO with clipped objectives can be defined as follows. In its implementation, we maintain two policy networks. The first one is the current policy $\pi_\theta(a_t|s_t)$ that we want to refine. The second is the policy that we last used to collect samples, denoted as $\pi_{\theta_{old}}(a_t|s_t)$. With the idea of importance sampling, we can evaluate a new policy with samples collected from an older policy, which can improve sample efficiency, denoted as follows.

$$\max_\theta \mathbb{E}_t \left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \widehat{A_t} \right] \tag{17}$$

With clipped objective, we compute a ratio between the new policy and the old policy:

$$r_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t) \tag{18}$$

This ratio measures how the difference between the two policies. We construct a new objective function to clip the estimated advantage function if the new policy is far away from the old policy. Our new objective function for k-th iteration becomes:

$$\mathcal{L}^{CLIP}_{\theta_k}(\theta) = \underset{\tau\sim\pi_k}{E}\left[\sum_{t=0}^{T}\left[\min\left(r_t(\theta)\widehat{A}^{\pi_k}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\widehat{A}^{\pi_k}_t\right)\right]\right] \tag{19}$$

If the probability ratio between the new policy and the old policy falls outside the range $[1-\epsilon, 1+\epsilon]$, the advantage function will be clipped.

A.9.2 Contrastive Representation Learning . Representation Learning is to learn representations that identify and disentangle useful information hidden in data when solving downstream tasks [2]. Contrastive representation learning uses the contrastive learning [3] method to learn representations that obey similarity constraints. It is often best understood as performing a dictionary look-up: a query should be similar to the keys it matches, but no other keys. If query and key are data augmentations of the same instance, they form a positive pair, otherwise a negative pair. Learning is formulated as minimizing a contrastive loss by measuring the similarity of sample pairs in a representation space.

There are some works utilizing contrastive learning to extract compressed representations from raw inputs in RL. CURL [2] adopts contrastive representation learning and data augmentation for sample-efficient RL for image-based control. LESSON [4] and HESS [5] adopt triplet loss [6] to train representations with slow dynamics. CACL [7] uses the supervised contrastive learning [8] method to align message spaces for communication. COLA [9] uses the supervised contrastive learning method to learn discrete representations of common knowledge.

A.10 Detailed Explanations about Baselines

In this section, we want to answer two questions about the baselines.

(1). Why do we choose these baselines?

In this paper, we select the state-of-the-art (SOTA) algorithm PPO [38] and SAC [14], the tree-based planning algorithm SGT [17], and the context-based RL method CORRO [46] as baselines to verify the effectiveness of our proposed method.

- PPO is a popular deep reinforcement learning algorithm for solving discrete and continuous control problems, which is an on-policy algorithm.
- SAC is an off-policy actor-critic deep RL algorithm based on the maximum entropy reinforcement learning framework.
- SGT adopts the divide-and-conquer style that recursively predicts intermediate sub-goals between the start state and goal. It can be interpreted as a binary tree that is most similar to our settings.
- CORRO is based on the framework of context-based meta-RL, which adopts a transition encoder to extract the latent context as a policy condition.

(2). How to guarantee the fairness of experiments?

The comparision is fair, because CORRO is a baseline that is similar to our setting that uses other datasets. As described in Section B of Algorithm 1 in the paper of CORRO [10], we **pre-train its transition encoder** $E_{\{}\theta_1\}$ **using all of our collected subtask transitions.**

Different from the procedure of making data augmentation, we regard other subtasks as negative instances, and apply the contrastive objective in Equation (8) defined in CORRO to train the transition encoder.

Applied the dataset collected by us to CORRO and ours, our model outperforms CORRO in most of the cases, which indicates the strong inference capacity of our designed framework.

B REFERENCES FOR APPENDIX

[1]. Schulman J, Wolski F, Dhariwal P, et al. Proximal policy optimization algorithms[J]. arXiv preprint arXiv:1707.06347, 2017.
[2]. Bengio Y, Courville A, Vincent P. Representation learning: A review and new perspectives[J]. IEEE transactions on pattern analysis and machine intelligence, 2013, 35(8): 1798-1828.
[3]. Hadsell R, Chopra S, LeCun Y. Dimensionality reduction by learning an invariant mapping[C]//2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06). IEEE, 2006, 2: 1735-1742.
[4]. Li S, Zheng L, Wang J, et al. Learning subgoal representations with slow dynamics[C]//International Conference on Learning Representations. 2020.
[5]. Li S, Zhang J, Wang J, et al. Active hierarchical exploration with stable subgoal representation learning[J]. arXiv preprint arXiv:2105.14750, 2021.
[6]. Lo Y L, Sengupta B. Learning to Ground Decentralized Multi-Agent Communication with Contrastive Learning[J]. arXiv preprint arXiv:2203.03344, 2022. [7]. Khosla P, Teterwak P, Wang C, et al. Supervised contrastive learning[J]. Advances in Neural Information Processing Systems, 2020, 33: 18661-18673.
[8]. Xu Z, Zhang B, Li D, et al. Consensus Learning for Cooperative Multi-Agent Reinforcement Learning[J]. arXiv preprint arXiv:2206.02583, 2022.
[9]. Chen T, Kornblith S, Norouzi M, et al. A simple framework for contrastive learning of visual representations[C]//International conference on machine learning. PMLR, 2020: 1597-1607.

[10]. Yuan H, Lu Z. Robust task representations for offline meta-reinforcement learning via contrastive learning[C]//International Conference on Machine Learning. PMLR, 2022: 25747-25759.