A APPENDIX

The supplementary material provides additional results, discussions, and implementation details.

- In Section A.1, we detail the BabyAI environment and the chosen testing tasks.
- In Section A.2, we provide the implementation and training details for the PAE and the baseline algorithms.
- In Section A.3, we describe our PAE algorithm and the baseline algorithms.
- In Section A.4, we present additional experimental results and analysis.

Our code is included in the supplementary material for reproduction.

A.1 ENVIRONMENT AND TASK DETAILS

BabyAI and Minigrid The BabyAI platform enables research in grounded language learning involving humans. In BabyAI, agents maximize rewards by completing tasks within limited steps, guided by language instructions. The platform utilizes a grid world environment (Minigrid), a partially observable 2D grid world housing agents and objects (available in 6 colors): boxes, balls, doors, and keys. These entities occupy $N \times M$ tiled rooms interconnected by doors, which may be locked or closed. Agents can pick up, drop, and move objects, while doors require color-matching keys for unlocking. As shown in Figure 6, we evaluated our approach in the BabyAI across six environments spanning two task types: the Key Corridor task (KEYCORRS3R3, KEYCORRS4R3, KEYCORRS5R3) and the Obstructed Maze task (OBSTRMAZE1DL, OBSTRMAZE2DLHB, OBSTRMAZE1Q). Table 2 illustrates key properties of the Minigrid environment. Table 3 shows the 66 categories of templates provided by BabyAI, totaling 652 pieces of language knowledge.

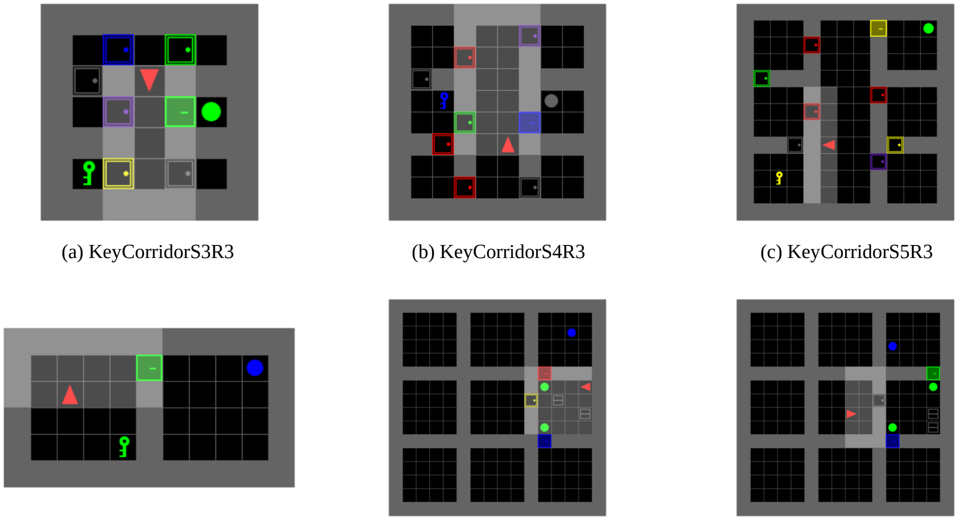

(a) KeyCorridorS3R3
(b) KeyCorridorS4R3
(c) KeyCorridorS5R3
(d) ObstructedMaze-1Dl
(e) ObstructedMaze-2D1hb
(f) ObstructedMaze-1Q

Figure 6: Six challenging environments we used to evaluate PAE.

Key Corridor In the Key Corridor task, the 'S' in the environment name's suffix indicates the room's size, while 'R' signifies the number of rows. The agent must retrieve an item behind a locked door. The key, concealed in a different room, must be found by the agent exploring the environment. Figure 10 illustrates the flow chart for the agent to complete the Key Corridor tasks.

Obstructed Maze In the Obstructed Maze task, a small blue ball is concealed in a corner. With the door locked and blocked by the ball, the key to unlock it is hidden inside a box. The agent must execute a series of precise steps. Typically, this involves opening the box to obtain the correctly colored key, using the key to unlock the door, and subsequently accessing the door to reach the objective. Figure 11 illustrates the flow chart for the agent to complete the Obstructed Maze tasks.

Table 2: Examples of BabyAI environments and their entity labeling.

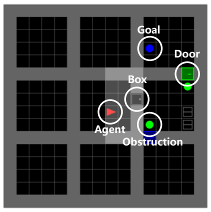
Illustration	**Observation**	**Other Properties**
	**Observation**: In BabyAI, the agent's observation space is an egocentric 7x7 grid representation oriented in the direction the agent faces. Each cell within this grid has three defining features: object, color, and state. These features identify the object in the cell, its color, and its state (e.g., distinguishing between a locked and unlocked door).	**Action**: Turn left, Turn right, Move forward, Pick up an object, Drop, Toggle, Done **Reward**: A reward of ‘1 - 0.9 × (step_count / max_steps)’ is given for success and 0 for failure. **Termination**: The agent picks up the correct object. Timeout.

Table 3: All knowledge is provided by BabyAI and covers 66 template categories totaling 652 entries. [C] is one of 6 possible colors: green, grey, yellow, blue, purple, and red.

Go to the <Object>	Open the <Object>	Pick Up the <Object>
go to the ball		
go to the box		**pick up** the ball
go to the door	**open** the box	**pick up** the box
go to the key	**open** the door	**pick up** the key
go to the [C] ball	**open** the [C] box	**pick up** the [C] ball
go to the [C] box	**open** the [C] door	**pick up** the [C] box
go to the [C] door		**pick up** the [C] key
go to the [C] key		

Put the <Object> Next to the <Object>		
put the ball **next to** the ball	**put the** ball **next to** the [C] ball	
put the ball **next to** the box	**put the** ball **next to** the [C] box	
put the ball **next to** the door	**put the** ball **next to** the [C] door	**put the** [C] box **next to** the [C] ball
put the ball **next to** the key	**put the** ball **next to** the [C] key	**put the** [C] box **next to** the [C] box
put the box **next to** the ball	**put the** box **next to** the [C] ball	**put the** [C] box **next to** the [C] door
put the box **next to** the box	**put the** box **next to** the [C] box	**put the** [C] box **next to** the [C] key
put the box **next to** the door	**put the** box **next to** the [C] door	**put the** [C] key **next to** the [C] ball
put the box **next to** the key	**put the** box **next to** the [C] key	**put the** [C] key **next to** the [C] box
put the key **next to** the ball	**put the** key **next to** the [C] ball	**put the** [C] key **next to** the [C] door
put the key **next to** the box	**put the** key **next to** the [C] box	**put the** [C] key **next to** the [C] key
put the key **next to** the door	**put the** key **next to** the [C] door	
put the key **next to** the key	**put the** key **next to** the [C] key	

A.2 IMPLEMENTATION DETAILS

A.2.1 MODEL ARCHITECTURE DETAILS OF PLANNER

This section provides more details of the Planner architecture used in our method. Planner consists of three components, i.e., encoding, alignment, and forwarding of states and knowledge.

Encoding Following BabyAI, the environment observation is a symbolic $H \times W \times 3$ fully observed representation, with N varying according to the environment's size. Each cell in the $H \times W$ grid has 3 features indicating the object's type (e.g., boxes, balls, doors, keys), color (from 6 possible choices), and its state (like open or closed for doors). The Planner embeds these features into type/color/state embeddings with dimensions of 5, 3, and 2, resulting in a visual embedding s_0 of size $HW \times 10$. To process s_0, the Planner employs a 4-layer shape-preserving convolution neural network interleaved with Exponential Linear Units, where each convolution layer has 16 filters with the size of 3×3, stride of 1, and padding of 1. The convolution neural network's output $\hat{s}_t$ provides an embedding layer sized $HW \times 16$. The Planner integrates a randomly initialized, trainable position embedding $\mathbf{E}_{pos}$ with $\text{Conv}(s_0)$ to encapsulate the state's content and spatial features and make preparations for alignment.

Meanwhile, the Planner utilized a pre-trained BERT model with frozen parameters to understand the semantics and encode knowledge. Specifically, we use the vector output of the encoder (in this case, BERT) in the [CLS] position for the sentence embedding. This model encoded a set of n natural language instructions, supplied by the Oracle, into instruction embeddings with dimensions of $n \times 768$. Subsequently, a linear projection layer transformed these embeddings to produce outputs with dimensions of $n \times 16$, denoting as $\hat{\mathbf{k}}$.

Alignment To integrate the two types of input embeddings, $\hat{\mathbf{s}}_0 \in \mathbb{R}^{HW \times 10}, \hat{\mathbf{k}} \in \mathbb{R}^{n \times 16}$, we employ the scaled dot-product cross-attention mechanism. The query Q is computed using $\hat{\mathbf{k}}W_Q$, and the key and value are derived from $\hat{\mathbf{s}}_t W_K$ and $\hat{\mathbf{s}}_t W_V$, respectively.

Forward After aligning knowledge and environment states, the cross-attention layer yields an output, $\hat{\mathbf{k}}_{s_0}$, which is a $n \times 16$ context vector capturing the correlation between the knowledge set and the current state. We project this context vector onto n-dimensional logits through a linear layer and finally get the most suitable external knowledge for the Actor after softmax sampling.

A.2.2 MODEL ARCHITECTURE DETAILS OF ACTOR

This section details the Actor architecture used in our approach. The Actor consists of a policy network and a discriminative network.

The policy network adopts a similar network architecture as the Planner, with the difference that the inputs to the policy network at each time step t are a partial observation of size $7 \times 7 \times 3$ and a single piece of knowledge k filtered by the Planner. Due to the reduction of the information content of the input state, we remove the shape-preserving convolution and add the position information directly to the state embedding ${}^{p}s_t$, obtaining ${}^{p}\hat{s}_t$ of size $(7 \times 7) \times 10$. For the knowledge selected by the Planner, we utilize the same parameter-frozen BERT model to get the knowledge embedding $\hat{k}$ of dimension 64 after a linear projection. We compute the cross attention of ${}^{p}\hat{s}_t$ and $\hat{k}$ by using $\hat{k}_t$ to generate query Q, ${}^{p}\hat{s}_t$ for key K and value Q. Then, we used four strided convolution layers to extract the state information to get $1 \times 1 \times 32$ state embedding. Finally, we concatenated state embedding and knowledge-state alignment embedding and passed it through one linear layer and softmax layer to get the logit distribution over the seven actions.

The Actor uses a separate discriminative network $q_\phi(k|{}^{p}s_t)$ to infer the knowledge provided by the Planner from the learned strategies. Intuitively, the Actor is motivated to bridge the connection between knowledge and states and explore states with a close relationship with k. The discriminative network $q_\phi(k|{}^{p}s_t)$ is optimized using a cross-entropy loss $\mathcal{L}_\phi$, denoting as:

$$\mathcal{L}_\phi({}^{p}s_t, k) = -\sum_{k' \in \mathcal{K}} (k' = k) \cdot \log(q_\phi(k'|{}^{p}s_t)) \tag{12}$$

Specifically, we employ a same-structure strided convolution network in the policy network to get the state feature ${}^{p}\tilde{s}_t$, and knowledge embeddings $\hat{\mathbf{k}}$ similar to the Planner:

$$\begin{aligned} \tilde{s}_t &= \text{Conv}({}^{p}s_t) && \in \mathbb{R}^{1 \times 32} \\ \hat{\mathbf{k}}_a &= \text{Proj}(\text{LM}([k^{(1)}, k^{(2)}, \ldots, k^{(n)}])) && \in \mathbb{R}^{n \times 32} \end{aligned} \tag{13}$$

Then a dot-product operation between ${}^{p}\tilde{s}_t$ and knowledge embeddings is performed, followed by a softmax layer outputting $q_\phi(k|{}^{p}s_t)$:

$$q_\phi(k|{}^{p}s_t) = \text{SoftMax}(\hat{\mathbf{k}}_a \cdot \tilde{{}^{p}s_t}^{T}) \tag{14}$$

Interestingly, whereas the discriminative network was originally derived from the definition of mutual information, we find that discriminative network can be viewed as predicting and modeling the knowledge distribution of the planner from its own perspective in order to strengthen the connection between state and knowledge.

A.2.3 TRAINING DETAILS

Each model was trained using five independent seeds on a system with 112 Intel® Xeon® Platinum 8280 cores and 6 Nvidia RTX 3090 GPUs. Run times ranged from 10 hours (for OBSTRUCTED-MAZE1DL) to 100 hours (for the longest KEYCORRIDORS5R3 task).

A.2.4 HYPERPARAMETERS

For PAE, we ran a grid search over batch size $\in \{8, 32, 150\}$, unroll length $\in \{20, 40, 100, 200\}$, entropy cost for the Actor $\in \{0.0001, 0.0005, 0.001\}$, the Actor learning rate $\in \{0.0001, 0.0005, 0.001\}$, the Planner learning rate $\in \{0.0001, 0.0005, 0.001\}$, entropy cost for the Planner $\in \{0.001, 0.005, 0.01\}$. Table 4 shows the best parameters obtained from the search.

For RND and ICM, we followed previous work (Raileanu & Rocktäschel, 2020) and used batch size of 32, unroll length of 100, RMSProp optimizer learning rate of 0.0001 with $\epsilon = 0.01$ and momentum of 0, intrinsic reward coefficient of 0.1, and entropy coefficient of 0.0005, which were the best values they found using grid searches in the same tasks.

For AMIGo and L-AMIGo, we followed previous work (Campero et al., 2020; Mu et al., 2022) and used teacher policy batch size of 32, student policy batch size of 32, teacher grounder batch size of 100, RMSProp optimizer learning rate of 0.0001 with $\epsilon = 0.01$ and momentum of 0, intrinsic reward coefficient of 1, unroll length of 100, value loss cost of 0.5, entropy cost of 0.0005, which were the best values for the same tasks described in their paper.

Table 4: Hyperparameters of PAE adopted in all the experiments.

Parameter	Value
Actor batch size B	32
Planner batch size B_p	32
Unroll Length	100
Discount	0.99
Value loss cost	0.5
Actor entropy cost	0.0005
Planner entropy cost	0.5
Actor mutual information cost α	0.0001
Actor cross attention head number	1
Planner cross attention head number	1
Pretrained language model	Bert-base
Pretrained language model output dim	768
Actor language embedding dim	64
Planner language embedding dim	256
Planner embedding projection dim	16
Actor learning rate	5e-4
Planner learning rate	5e-4
Adam betas	(0.9, 0.999)
Adam ϵ	1e-8
Clip gradient norm	40.0

A.3 ALGORITHMS DETAILS

A.3.1 ALGORITHMS DETAILS OF THE PAE

Algorithm 1 illustrates the overall rollout process of PAE. We implemented the PAE and reproduction baseline algorithms using TorchBeast[2] (Küttler et al., 2019), an IMPALA-based PyTorch platform for rapid asynchronous parallel training in reinforcement learning.

Algorithm 1: Asynchronous learning algorithm for PAE

```
Input: Buffer $\mathbf{B}$, Actor batch $\mathbb{B}$, Planner batch $\mathbb{B}_p$, Actor batch size $B$, Planner batch size $B_p$
Initialize: $\mathbf{B} \leftarrow \varnothing$, $\mathbb{B} \leftarrow \varnothing$, $\mathbb{B}_p \leftarrow \varnothing$, $\pi_a(a_t|s_t, k; \boldsymbol{\omega}) \leftarrow \boldsymbol{\omega}_0$, $\pi_p(k|s_0, \mathcal{K}; \boldsymbol{\theta}) \leftarrow \boldsymbol{\theta}_0$
Function INTERACT(Actor Policy: $\pi_a^*$, Planner Policy: $\pi_p^*$, Buffer: $\mathbf{B}$):
    $\{s_0, {}^p s_0\} \leftarrow$ Env.Reset(0)    ▷ $s_0$ denotes full observation, ${}^p s_0$ denotes partial observation
    $k \leftarrow \pi_p^*(k_0|s_0, \mathcal{K}; \boldsymbol{\theta}^*)$
    $k_{steps} \leftarrow 0$
    while True do
        $a_t \leftarrow \pi_a^*(a_t|{}^p s_t, k; \boldsymbol{\omega}^*)$
        $\{s_{t+1}, {}^p s_{t+1}, r_t, k_{done}, s_{done}\} \leftarrow$ Env.Step($a_t$)    ▷ Interact with the environment
        $k_{steps} \leftarrow k_{steps} + 1$
        if $s_{done}$ then
            $\{s_0, {}^p s_0\} \leftarrow$ Env.Reset(0)    ▷ This episode is finished
            $k \leftarrow \pi_p^*(k_0|s_0, \mathcal{K}; \boldsymbol{\theta}^*)$
            $k_{steps} \leftarrow 0$
        end
        if $k_{done}$ then
            $k \leftarrow \pi_p^*(k|s_0, \mathcal{K}; \boldsymbol{\theta}^*)$    ▷ The Actor successfully followed the Planner's guidance
            $k_{steps} \leftarrow 0$
        end
        Update $\mathbf{B}$    ▷ Update Buffer $\mathbf{B}$ using newly collected experience
    end
end
Function MAIN():
    $\pi_a^* \leftarrow \pi_a.copy()$    ▷ Make a parameter-fixed copy of $\pi_a$ for environment interacting
    $\pi_p^* \leftarrow \pi_p.copy()$    ▷ Make a parameter-fixed copy of $\pi_p$ for environment interacting
    $Th \leftarrow$ Thread(INTERACT, $\pi_a^*$, $\pi_p^*$, $\mathbf{B}$)    ▷ Create a thread for environment interacting
    while not converged do
        Update $\mathbb{B} \leftarrow \mathbf{B}$    ▷ Sample batch $\mathbb{B}$ of size $B$ from Buffer $\mathbf{B}$
        Train $\pi_a(a_t|{}^p s_t, k; \boldsymbol{\omega})$ on $\mathbb{B}$    ▷ Training policy $\pi_a$ using reinforcement learning
        Train $q_\phi(k|{}^p s_t)$ on $\mathbb{B}$    ▷ Training discriminative network $q_\phi$ using supervised learning
        Update $\pi_a^* \leftarrow \pi_a.copy()$    ▷ Update $\pi_a^*$ with latest model $\pi_a$
        Update $\mathbb{B}_p$ with $\{s_0, k, k_{steps}, s_{done}\}$ from $\mathbb{B}_p$
        if $|\mathbb{B}_p| > B_p$ then
            Train $\pi_p(k|s_0, \mathcal{K}; \boldsymbol{\theta})$ on $\mathbb{B}_p$    ▷ Training policy $\pi_p$ using reinforcement learning
            $\mathbb{B}_p \leftarrow \varnothing$
            Update $\pi_p^* \leftarrow \pi_p.copy()$    ▷ Update $\pi_p^*$ with latest model $\pi_p$
        end
    end
    $Th.join()$    ▷ Stop interacting with the environment
end
```

---

[2] `https://github.com/facebookresearch/torchbeast`

A.3.2 DETAILS OF THE BASELINE ALGORITHMS

Below, we describe the implementation details of the various baseline algorithms. **IMPALA** is an off-policy actor-critic framework. The actor-critic agent maintains a policy $\pi_\theta(a|x)$ and a value function $v_\theta(x)$, both parameterized by θ. Both the policy and value functions are refined using the actor-critic update rule. IMPALA also incorporates an entropy regularization loss. The update is derived from the gradient of a specific pseudo-loss function:

$$\begin{aligned}
&\mathcal{L}_{\text{Value}}(\theta)=\sum_{s\in\mathrm{T}}\left(v_s-V_\theta\left(x_s\right)\right)^2\\
&\mathcal{L}_{\text{Policy}}(\theta)=-\sum_{s\in\mathrm{T}}\rho_s\log\pi_\theta\left(a_s|x_s\right)\left(r_s+\gamma v_{s+1}-V_\theta\left(x_s\right)\right)\\
&\mathcal{L}_{\text{Entropy}}(\theta)=-\sum_{s\in\mathrm{T}}\sum_{a}\pi_\theta\left(a|x_s\right)\log\pi_\theta\left(a|x_s\right)\\
&\mathcal{L}(\theta)=g_v\mathcal{L}_{\text{Value}}(\theta)+g_p\mathcal{L}_{\text{Policy}}(\theta)+g_e\mathcal{L}_{\text{Entropy}}(\theta).
\end{aligned}\tag{15}$$

This decoupled architecture facilitates high throughput. Nonetheless, there can be a bias since the policy generating the trajectory might lag several updates behind the learner's policy when computing the gradient. IMPALA addresses this potential bias by employing the V-trace, which balances the variance-contraction trade-off in these off-policy updates.

RND The key idea of RND is to calculate intrinsic reward based on the prediction error linked to an agent's transfer. RND employs two neural networks: a fixed, randomly initialized target network that defines the prediction task and a predictor network trained on the agent's collected data. RND considers the prediction error from the predictor network, based on features produced by the target network, as a reward for novel states. We followed AMIGo (Campero et al., 2020), using a re-implemented version based on TorchBeast (Küttler et al., 2019) provided by (Raileanu & Rocktäschel, 2020).

ICM utilizes curiosity as an intrinsic reward, defining it as the agent's error in predicting its own behavior's consequences within a feature space learned through a self-supervised inverse dynamics model. Specifically, ICM encodes states s_t and s_{t+1} as features $\phi(s_t)$ and $\phi(s_{t+1})$, respectively, trained to predict a_t (i.e., the inverse dynamics model). The forward model takes $\phi(s_t)$ and a_t as inputs and predicts the feature representation $\phi(s_{t+1})$ for s_{t+1}. The prediction error within the feature space serves as a curiosity-driven intrinsic reward. We followed AMIGo (Campero et al., 2020) using a re-implemented version of ICM provided by (Raileanu & Rocktäschel, 2020). based on TorchBeast (Küttler et al., 2019).

AMIGo is a meta-learning method that automatically learns to self-propose adversarial motivational intrinsic goals. AMIGo consists of two subsystems: a goal-conditioned student policy that outputs the agent's actions in the environment and a goal-generating teacher that guides the student's training. These two components train in adversarial training, where the student maximizes rewards by reaching the goal as quickly as possible, and the teacher maximizes rewards by proposing goals that the student can reach but not too quickly. We use the open-source AMIGo (Campero et al., 2020) code[3] as a baseline for our PAE method.

L-AMIGo utilizes language as a generalized medium to emphasize relevant environmental abstractions. It is an extension of AMIGo where the teacher presents goals using language instead of coordinates (x, y). The student is a conditional policy, receiving language goals g_t instead of (x, y). Specifically, the student network is decomposed into policy and grounding networks. The policy network produces a distribution of goals based on the student's state, while the grounding network predicts the probability of achieving the goal.

---

[3] `https://github.com/facebookresearch/adversarially-motivated-intrinsic-goals`

A.4 ADDITIONAL RESULTS

A.4.1 QUANTITATIVE RESULTS

Figure 4 shows the training curves of our PAE method alongside the IMPALA, RND, ICM, AMIGo, and L-AMIGo baseline algorithms. Each model was trained using five independent random seeds. The results reported for each baseline and each environment are the best performance configuration of the policy for that environment. Table 5 gives the quantitative results of these comparison experiments and displays the mean extrinsic rewards and the number of steps (in millions) needed for each model to converge. Each entry consists of two rows of results, with the top row being the average extrinsic reward at the end of training and the bottom row being the minimal stable steps to attain that reward. Lower bottom row values signify quicker convergence, and "$> x$" indicates a lack of convergence within the maximum training steps "x".

Table 5: Comparison of PAE and baseline methods.

	Key Corridor Tasks			**Obstructed Maze Tasks**		
Model	**KEYCORRS3R3**	**KEYCORRS4R3**	**KEYCORRS5R3**	**OBSTRMAZE1DL**	**OBSTRMAZE2DLHB**	**OBSTRMAZE1Q**
PAE (Ours)	**0.89 ± 0.002** **6M**	**0.92 ± 0.005** **30M**	**0.94 ± 0.001** **90M**	**0.93 ± 0.004** **6M**	**0.87 ± 0.018** **150M**	**0.89 ± 0.006** **150M**
L-AMIGo	0.86 ± 0.040 12M	0.90 ± 0.011 60M	0.93 ± 0.016 160M	0.90 ± 0.030 20M	0.23 ± 0.403 >200M	0.47 ± 0.029 >300M
AMIGo	0.43 ± 0.246 >20M	0.22 ± 0.343 >60M	0.43 ± 0.599 >200M	0.45 ± 0.366 >20M	0.10 ± 0.175 >200M	0.18 ± 0.023 >300M
ICM	0.04 ± 0.052 >20M	0.30 ± 0.511 >60M	0.00 ± 0.000 >200M	0.31 ± 0.542 15M	0.00 ± 0.000 >200M	0.00 ± 0.000 >300M
IMPALA	0.00 ± 0.000 >20M	0.00 ± 0.000 >60M	0.00 ± 0.000 >200M	0.00 ± 0.006 >20M	0.00 ± 0.000 >200M	0.00 ± 0.000 >300M
RND	0.00 ± 0.000 >20M	0.00 ± 0.000 >60M	0.00 ± 0.000 >200M	0.00 ± 0.000 >20M	0.00 ± 0.000 >200M	0.00 ± 0.000 >300M

A.4.2 ABLATION EXPERIENTIAL RESULTS

Figure 7 illustrates the effect of introducing external knowledge on PAE performance. **Full-Model** is the full version of PAE. **w/o Curriculum** provides randomized guidance to the Actors through external knowledge, while **w/o Planner** eliminates the Planner's guidance.

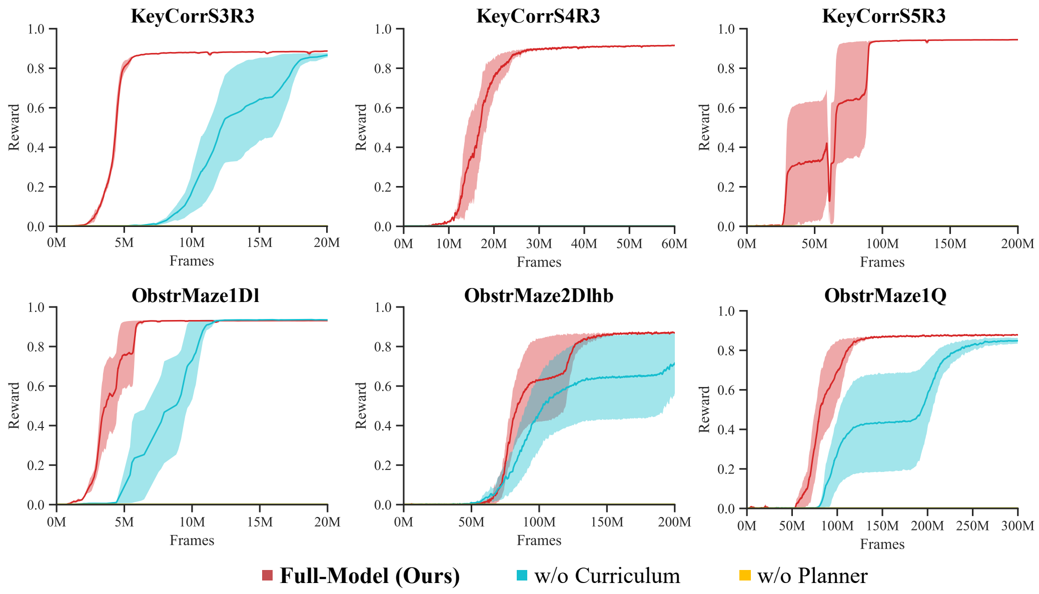


Figure 7: Ablation of PAE's external knowledge.

A.4.3 QUALITATIVE RESULTS

Figure 8 shows an example of PAE completing a task in KeyCorrS5R3. In this example, the Actor first needs to explore the environment to find the key, then open the door of the corresponding color, drop the key, and retrieve the item behind the door. Figure 9 shows the flow of the Planner forming an automatic curriculum in *KeyCorrS5R3* and the progressive acquisition of skills by the Actor. Typically, in the early stages, the Planner provides easy guidance. As the training progresses and the Actor's ability increases, the Planner provides more hard guidance. We can see in the trajectory that the Planner and Actor are progressing together and eventually complete the task successfully.

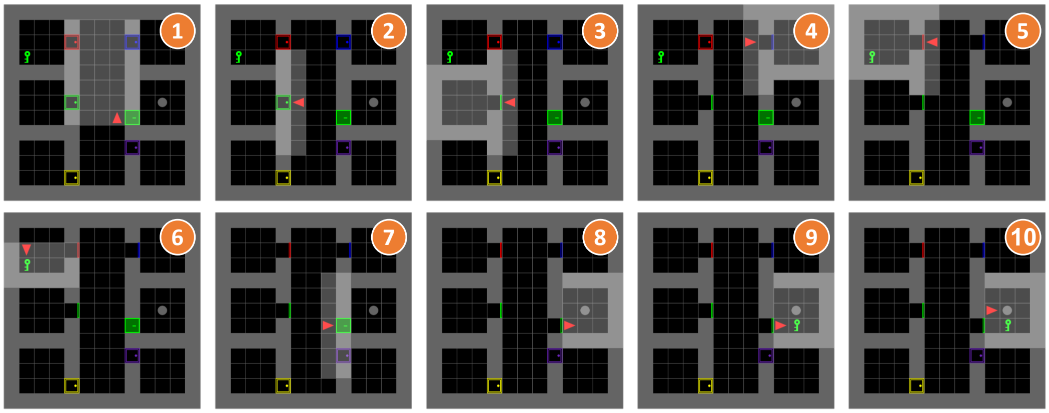


Figure 8: Example of PAE in completing a task in KeyCorrS5R3. To save space, we only sampled one sub-trajectory for display.

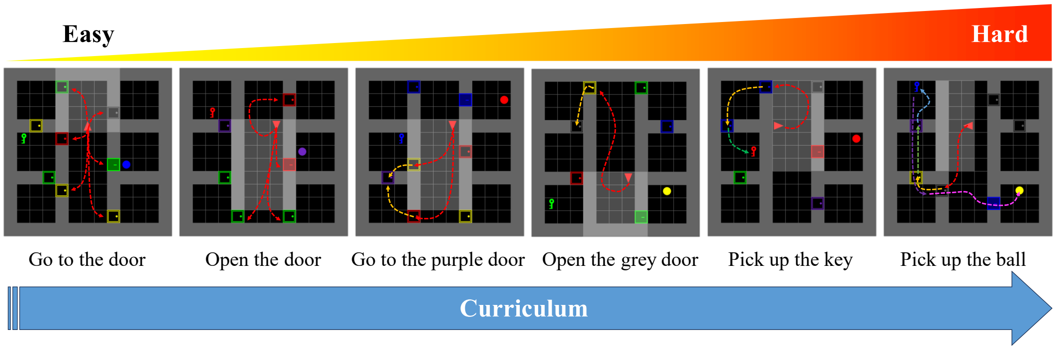


Figure 9: An illustration of the training process. The Planner forms an automatic curriculum as the training progresses, and the Actor progressively masters the skills.

A.5 FLOW CHART

Figure 10 and Figure 11 show flowcharts of the agent completing the Key Corridor task and the Obstructed Maze task, respectively.

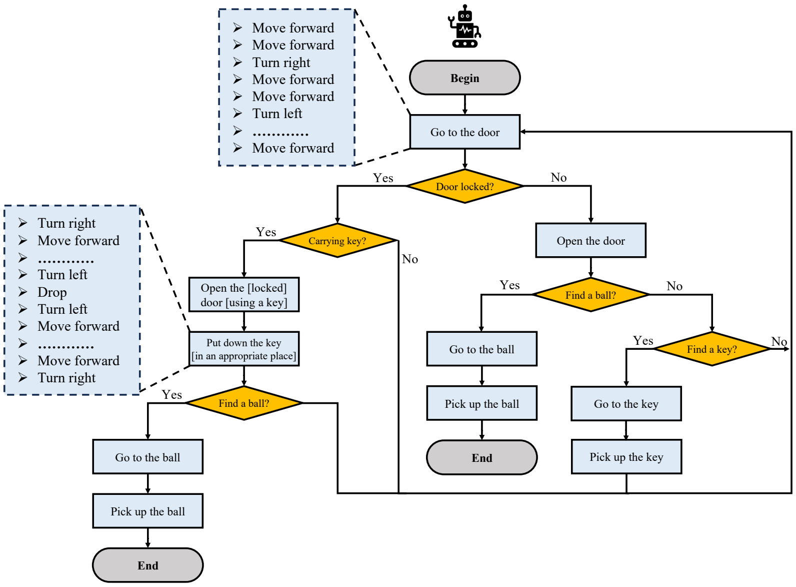


Figure 10: Flow chart of an agent completing Key Corridor tasks.

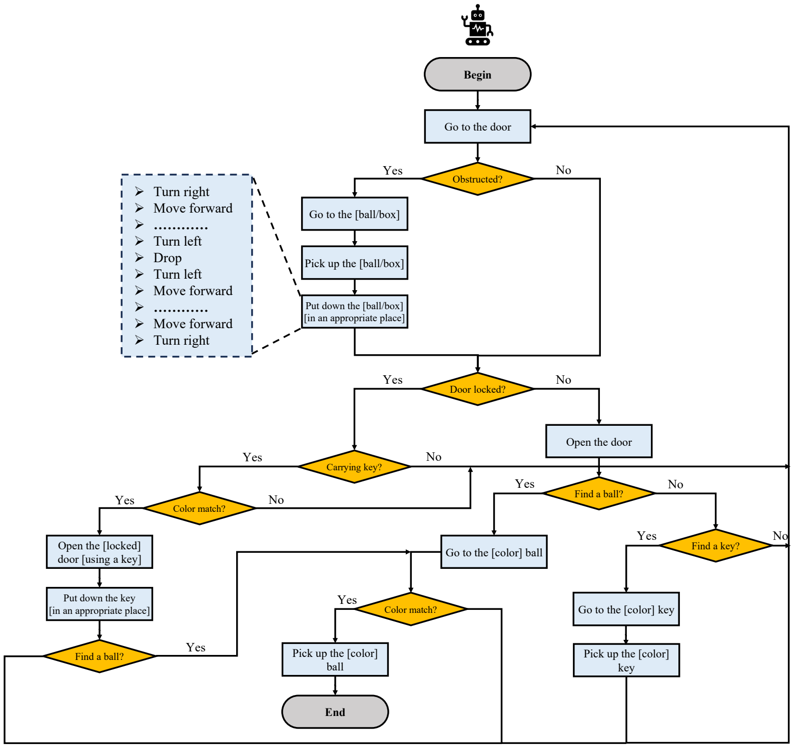


Figure 11: Flow chart of an agent completing Obstructed Maze tasks.