Natural Language-conditioned Reinforcement Learning with Inside-out Task Language Development and Translation

Jing-Cheng Pang [*1,2], Xin-Yu Yang [*1,2], Si-Hang Yang[1,2], and Yang Yu[†1,2]

[1]National Key Laboratory for Novel Software Technology, Nanjing University
[2]Polixir Technologies

Abstract

Natural Language-conditioned reinforcement learning (RL) enables the agents to follow human instructions. Previous approaches generally implemented language-conditioned RL by providing human instructions in natural language (NL) and training a following policy. In this *outside-in* approach, the policy needs to comprehend the NL and manage the task simultaneously. However, the unbounded NL examples often bring much extra complexity for solving concrete RL tasks, which can distract policy learning from completing the task. To ease the learning burden of the policy, we investigate an *inside-out* scheme for natural language-conditioned RL by developing a task language (TL) that is task-related and unique. The TL is used in RL to achieve highly efficient and effective policy training. Besides, a translator is trained to translate NL into TL. We implement this scheme as TALAR (**TA**sk **L**anguage with predic**A**te **R**epresentation) that learns multiple predicates to model object relationships as the TL. Experiments indicate that TALAR not only better comprehends NL instructions but also leads to a better instruction-following policy that improves 13.4% success rate and adapts to unseen expressions of NL instruction. The TL can also be an effective task abstraction, naturally compatible with hierarchical RL.

1 Introduction

Enabling the robot to work with humans is a hallmark of machine intelligence. Language is a vital connection between humans and machines [Kozierok et al., 2021], and it has been investigated for instructing robot execution, designing rewards, and serving as observation or action in reinforcement learning (RL) [Luketina et al., 2019]. We are especially interested in developing agents that follow human instructions in this broad context. Natural language-conditioned reinforcement learning (NLC-RL) is a promising tool in this pursuit, which provides the policy with human instructions in natural language (NL) and trains the policy with RL algorithms. In this *outside-in* learning (OIL, Figure 1-left) scheme, the policy is directly exposed to the NL instructions. Thus, the policy must comprehend the NL instructions and complete the RL tasks simultaneously.

However, natural language is an unbounded representation of human instruction, which imposes an additional burden on the policy when solving concrete RL tasks. For example, to ask a robot to bring a drink, one may say: *Get me a drink*, while another may ask: *Can you take the beverage to me?* Despite having identical meanings, the two NL instructions are expressed in vastly different ways. To complete human instructions, the policy must simultaneously comprehend these diverse, unbounded NL instructions and solve the RL tasks, resulting in inefficient policy learning.

In this paper, we investigate an *Inside-Out* Learning (IOL) scheme to enable efficient and effective policy learning in NLC-RL, as depicted in Figure 1-right. The IOL develops a task language (TL) that is task-related and uniquely represents human instruction. The TL can be utilized in RL to alleviate the burden of policy

[*] Equal contribution. Email: `pangjc@lamda.nju.edu.cn` and `yangxy@lamda.nju.edu.cn`
[†] Corresponding author. Email: `yuy@nju.edu.cn`

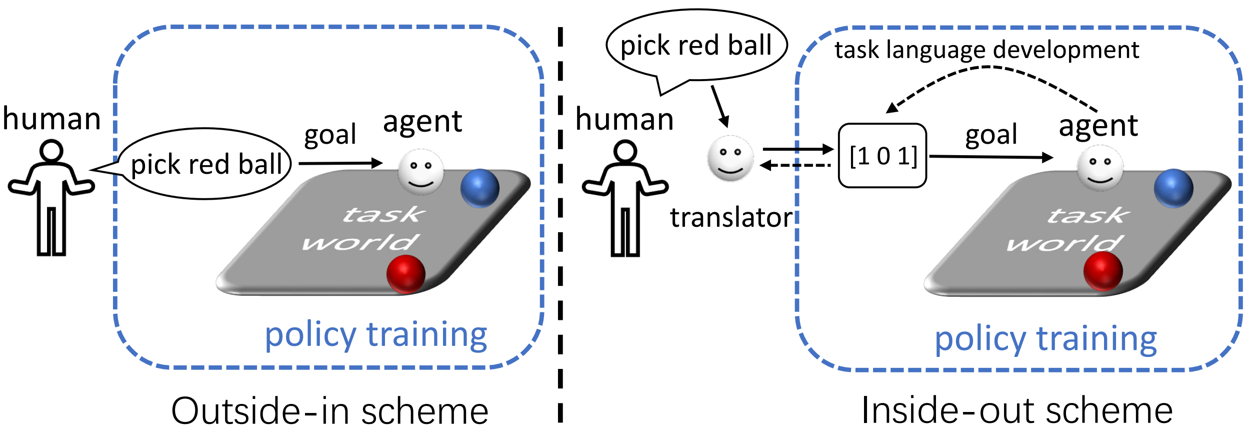


Figure 1: An illustration of OIL and IOL schemes in NLC-RL. **Left:** OIL directly exposes the NL instructions to the policy. **Right:** IOL develops a task language, which is task-related and a unique representation of NL instructions. The solid lines represent instruction following process, while the dashed lines represent TL development and translation.

learning by comprehending diverse NL expressions. In addition to developing TL, IOL trains a translator that translates NL into TL. A crucial aspect of IOL is how the task language is represented. We believe that *expressiveness* and *conciseness* are essential properties of language representation. Expressiveness ensures that the TL accurately reflects human instruction, while conciseness facilitates policy comprehension. To satisfy these two requirements, we propose representing TL with predicate representation, which is deemed expressive [Borgida, 1996] and concise as a discrete representation.

We introduce an implementation of IOL, called TALAR for **TA**sk **L**anguage with predic**A**te **R**epresentation. TALAR consists of three components: (1) a TL generator that generates TL with predicate representation, (2) a translator that translates NL into TL and (3) a policy that solves the RL tasks assigned by human instructions. Specifically, the TL generator develops TL through the identification of object relationships. To achieve this, the TL generator learns multiple (anonymous) predicates and their arguments to model the relationships between objects. The translator is trained to translate NL into TL using the variational auto-encoder [Kingma and Welling, 2014] algorithm. With the optimized translator, TALAR trains an instruction-following policy that completes human instructions.

Our contributions include the following: We propose a novel NLC-RL scheme, IOL, that enables highly efficient policy learning. IOL develops TL that serves as a unique representation of human instruction and trains the policy following TL. Besides, we present a specific IOL implementation, including a neural network architecture that automatically discovers relationships between the objects. Through our experiments in the CLEVR-Robot environment [Jiang et al., 2019], we find that TALAR can better translate diverse NL expressions into a unique representation compared to traditional OIL methods. A policy can learn to complete human instructions efficiently and adapt to unseen NL instructions with the resulting TL. Moreover, the resulting TL effectively represents human instruction, providing a solid baseline for abstraction in hierarchical RL [Barto and Mahadevan, 2003].

2 Related Work

This section begins with a summary of prior research on instruction following with RL, followed by two paragraphs discussing works pertinent to our methodology, i.e., language generation in RL and language translation.

Instruction following with RL. Instruction-following problems require agents to perform tasks specified by NL instructions. Previous methods employ RL to train an instruction-following policy and expose the NL directly to the policy. For example, [Hill et al., 2021] encodes a single-sentence instruction with a pre-trained language model and feeds the policy with the NL encoding. [Misra et al., 2018] learns a policy that maps NL instructions to action sequences by marginalizing implied goal locations. [Chaplot et al., 2018] combines human instructions with agent observations via a multiplication-based mechanism and then pre-trains the instruction-following policy using behaviour cloning [Pomerleau, 1991]. Instead of exposing NL to the policy, [Akakzia et al., 2021] encodes NL to a manually-designed binary vector in which each element has meaning. Besides, the instruction-following policy has close ties to Hierarchical RL [Barto and Mahadevan, 2003] because the instructions can be naturally viewed as a task abstraction for a low-level policy [Blukis et al., 2022]. HAL [Jiang et al., 2019] takes advantage of the compositional structure of NL and makes decisions directly at the NL level to solve long-term, complex RL tasks. These previous methods either expose the unbounded NL instructions directly to the policy or encode the NL instructions to a scenario-specific manual vector, both of which have limitations. In contrast to them, we propose developing a task-related task language that is a unique representation of NL instruction and is, therefore, easily understood by the policy.

Language representation in RL. We are interested in language representation, which is fundamental to developing task language. RL communities often consider language representation when agents learn effective message protocol to communicate with their partners [Eccles et al., 2019, Kang et al., 2020, Simões et al., 2019, Patel et al., 2021]. Motivated by the discrete nature of human language, discrete representation has been widely used in prior research. For example, [Li et al., 2022] enables agents to communicate using discrete messages and demonstrates that discrete representation has comparable performance to continuous representation with a much smaller vocabulary size. One-hot representation [Patel et al., 2021, Lazaridou et al., 2017] and binary representation [Akakzia et al., 2021, Oliehoek and Amato, 2016] are prevalent forms of discrete language representation. For instance, [Patel et al., 2021] uses one-hot language representation to allow two agents to communicate to differentiate between images. However, these representations only employ discrete symbols and operate on a propositional level, lacking the ontological commitment of the predicate representation that the world consists of objects and their relationships [Russell and Norvig, 1995]. In this paper, we develop the task language following the discrete form of language representation while the predication representation is used.

Language translation. In this paper, TALAR translates NL into TL, which lies in the domain of machine translation [Bahdanau et al., 2015]. In this field of study, numerous approaches have been developed [Rivera-Trigueros, 2022, Stahlberg, 2020]. Encoder-decoder [Cho et al., 2014] is a promising machine translation tool because of its ability to extract the effective features of the input sentences. For example, [Pagnoni et al., 2018] proposes to utilize a continuous latent variable as an efficient machine translation feature based on a variational auto-encoder [Kingma and Welling, 2014]. In this paper, we adhere to this class of machine translation techniques that employ an encoder-decoder structure, treating the NL as the source language and the TL as the target language.

3 Background

RL and NLC-RL. A typical RL task can be formulated as a Markov Decision Process (MDP) [Puterman, 1994, Sutton and Barto, 1998], which is described as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, r, \gamma, d_0)$. Here $\mathcal{S}$ represents the state space. $\mathcal{A}$ is the finite action space defined by $\mathcal{A} = \{a_0, a_1, \cdots, a_{|\mathcal{A}|-1}\}$. P represents the probability of transition while r represents the reward function. γ is the discount factor determining the weights of future rewards, whereas d_0 is the initial state distribution. A policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ is a mapping from state space to the probability space over action space. In NLC-RL, the agent receives an NL instruction (L) that reflects the human's instruction on the agent. An instruction-following policy $\pi(\cdot|s_t, L)$ is trained to make decisions based on the current state s_t and NL instruction L. The overall objective of NLC-RL is to maximize the expected return under different NL instructions:

$$J = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t|L) \big| s_0 \sim P, a_t \sim \pi(\cdot|s_t, L)\right]. \tag{1}$$

For the accuracy of sake, we use L_{N} and L_{T} to denote NL and TL, respectively.

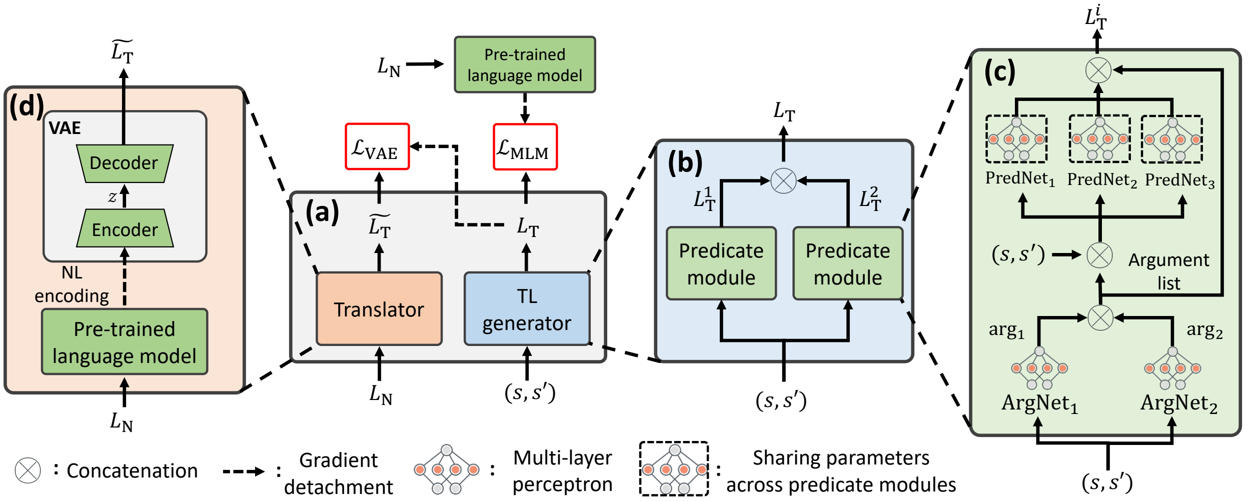


Figure 2: Overall training process of task language development and translation. **(a)** The overall training process. **(b)** Network architecture of the TL generator. **(c)** Architecture of one predicate module. **(d)** Network architecture of the translator. The number of predicate modules, arguments and predicate networks can be adjusted according to the task scale.

Predicate representation uses discrete binary vectors to represent the relationship between objects or the property of an individual object. For example, the predicate representation vector [1, 1, 0, 0, 0, 1, 0] could represent a predicate expression `Pred(a,b)`. In this instance, `Pred` is a predicate that represents a relationship, and the symbols `a` and `b` are its arguments. In the vector, the first code [1] in the vector indicates that the value of `Pred` is True (i.e., the relationship holds), whereas the red and blue one-hot codes represent the indexes of arguments `a` and `b`, respectively. It is demonstrated that predicate representation can be attained by employing networks to output predicate values. This way, the learning system can automatically identify relationships in a block stack task [Asai, 2019]. In TALAR, neural networks learn both predicates and their arguments. Refer to Appendix A for supplementary discussions on predicate representation.

Natural language as input for the neural network. NL sentences are in variable lengths and cannot be fed directly into a fully connected network. A standard solution is to encode each word as an embedding [Mikolov et al., 2013] and loop over each embedding using a recurrent model. Except for the recurrent model, Bert [Devlin et al., 2019] tokenizes the words before extracting sentence embedding features based on these tokens. Transformer [Vaswani et al., 2017] is the predominant model for natural language processing. Since we only need to convert NL sentences to fixed-length encoding in our experiments, we employ a pre-trained, lightweight Bert model.

4 Method

This section presents our TALAR method for efficient policy learning in NLC-RL, which is based on the IOL scheme. We begin by introducing the task dataset for task language development and translation.

Definition 1 (**Task dataset**). *A task dataset $\mathcal{D} = \{(s, s', L_\mathrm{N})_i\}$ consists of multiple triplets. Each triplet contains a natural language sentence L_N and a task state pair (s, s'), where L_N describes state change from s to s' in natural language.*

We use a state pair instead of a single state for the following reasons: (1) NL instruction frequently describes a state change, e.g., turn the wheel to the left; (2) it is not easy to describe a single state concisely in complex task scenarios. We let a person describe each state pair to collect the task dataset. Figure 2 illustrates how TALAR makes use of task dataset for task language development and translation. The subsequent subsections will elaborate on three critical components of TALAR: TL generation in predicate representation, NL translation by recovering TL, and policy training with reinforcement learning. Appendix C presents a summary of TALAR's training procedures.

4.1 TL Generation in Predicate Representation

TALAR trains a TL generator $g_\theta(s, s')$ represented by neural networks and parameterized with θ, which takes a state pair (s, s') as the input and outputs task language L_T. Next, we will introduce how task language is developed by describing the network structure of the TL generator and how to optimize the TL generator.

Network architecture of TL generator. As depicted in Figure 2(b-c), the TL generator comprises N_{pm} instances of predicate modules (PM). Each PM first extracts N_{a} arguments $(\text{arg}_1, \text{arg}_2, \cdots, \text{arg}_{N_\text{a}})$ according to the input state pair, and then determines the Boolean values of N_{pn} predicates, given the extracted argument list. The predicate values are concatenated with the argument list and form the task language L_{T}^i of the i-th module. Finally, the TL generator concatenates all PMs' output and generates the task language L_{T}. Note that the number of PMs N_{pm}, arguments N_{a}, and predicates N_{pn} can be modified based on the RL task scale.

Specifically, each PM extracts the arguments through argument networks, denoted by $\text{ArgNet}_i(s, s')$. An argument network is implemented as fully-connected networks ending with a Gumbel-Softmax activation layer [Jang et al., 2017]. Through the Gumbel-Softmax, the argument network is able to output a discrete one-hot vector arg_i in form like $(0, 1, \cdots, 0)$, which represents an anonymous object. TALAR utilizes multiple predicate networks, denoted by $\text{PredNet}_i(s, s', \text{arg}_1, \cdots, \text{arg}_{N_\text{a}})$, to determine the Boolean values of a set of anonymous predicates. Each predicate network outputs a 0-1 value, ending with a Gumbel-Softmax layer. All these 0-1 values are concatenated together with the argument list, yielding the task language L_{T}^i of the i-th PM. Note that without the argument list contained in L_{T}^i, the resulting language cannot express different objects and therefore loses its expressiveness.

All predicate networks within the same PM receive the identical argument list. In the TL generator, there are multiple PMs, each possessing its argument networks, while the parameters of each predicate network PredNet_i are shared across PMs. The parameter sharing here makes the $\textbf{PredNet}_i$ in different PMs identical, requiring them to capture consistent relations among the various arguments because they accept different arguments across PMs. Finally, the total task language is represented by $L_{\text{T}} = [L_{\text{T}}^1, \cdots, L_{\text{T}}^{N_{\text{pm}}}]$, which is a discrete binary vector. The Gumbel-Softmax activation technique permits the differentiation of the entire TL generation procedure.

Training of the TL generator. Training the TL generator ensures that the generated TL captures the crucial aspects of a given state transition. Based on this idea, TALAR uses the Masked Language Modeling (MLM) technique [Devlin et al., 2019] to train the TL generator. MLM masks L_{N} sentences at random. For example, when the original sentence is *It is a happy day*, the masked sentence could be *It* `Mask` *a happy* `Mask`. Then, MLM utilizes $L_{\text{T}} = g_\theta(s, s')$ and the masked L_{N} to predict the masked words. We implement the above process using a pre-trained Bert language model (LM), which first tokenizes L_{N} into tokens T. Then, TALAR selects two random token positions of T and replaces each with a unique token `Mask`. The TL generator is trained to optimize an MLM loss, which aims to predict the original tokens with masked tokens and task language:

$$\mathcal{L}_{\text{MLM}}(\theta) = \mathop{\mathbb{E}}_{(s,s',L_{\text{N}})\sim\mathcal{D}} \left[-\sum_{i\in M} \log f(T_i | b(T_{\setminus M}), g_\theta(s, s')) \right], \tag{2}$$

where M denotes the set of the masked positions, $T_{\setminus M}$ denotes the masked version of L_{N}'s tokens, T_i is the i-th token, b is the Bert model, and f is a fully-connected network. Note that f is also optimized via gradient backpropagation. We omit the notion of its parameters for simplicity.

4.2 NL Translation by Recovering TL

The objective of the translator is to translate the NL to the TL. TALAR trains the translator using variational auto-encoder (VAE) [Kingma and Welling, 2014]. Specifically, given a TL $L_{\mathrm{T}} = g_\theta(s, s')$ and corresponding NL L_{N}, we expect the VAE can recover L_{T} from L_{N}. Figure 2(d) presents the structure of the translator. TALAR uses a pre-trained LM to process L_{N} and a VAE to recover the task language. We let $\widetilde{L_{\mathrm{T}}}$ denote the TL generated by translator, q_{ϕ_1} the VAE encoder parameterized with ϕ_1, and p_{ϕ_2} the VAE decoder parameterized with ϕ_2. Then the VAE is trained to minimize the VAE loss:

$$L_{\mathrm{VAE}}(\phi_1, \phi_2) = \mathop{\mathbb{E}}_{(s,s',L_{\mathrm{N}})\sim\mathcal{D}} \left[D_{\mathrm{KL}}(q_{\phi_1}(z|L_N), p(z)) - \mathop{\mathbb{E}}_{z\sim q_{\phi_1}(\cdot|L_{\mathrm{N}})} \left[\log p_{\phi_2}(L_{\mathrm{T}}|z)\right] \right], \tag{3}$$

where $L_{\mathrm{T}} = g_\theta(s, s')$, $z \sim q_{\phi_1}(\cdot|L_{\mathrm{N}})$ is the encoding generated by VAE encoder, and D_{KL} stands for KL-divergence.

We choose VAE because of its capacity to learn effective latent features. However, VAE is not essential to implement IOL, as the translator can be trained using alternative supervised learning methods. We demonstrate the effectiveness of VAE empirically in Section 5.4. While some may be concerned about the translator's ability to recover L_{T} from L_{N} accurately, we emphasize that the translator is primarily responsible for recovering the key positions that reflect the NL instructions and not the entire L_T. With the optimized translator, an instruction-following policy is trained to complete the human instructions, as described below.

4.3 Policy Training With Reinforcement Learning

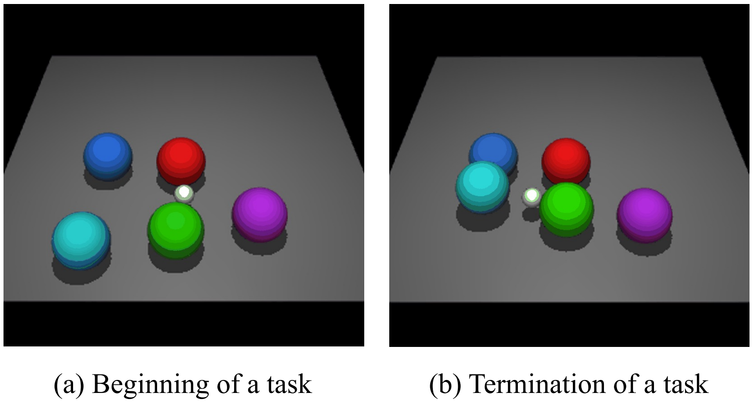
TALAR uses reinforcement learning to train an Instruction-Following Policy (IFP) $\pi(\cdot|s, \widetilde{L_{\mathrm{T}}})$. When the agent collects samples from the environment, the task generates a random human instruction in NL, which is then translated into the task language $\widetilde{L_{\mathrm{T}}}$ by the translator. Next, the IFP makes decisions for the entire episode based on the current observation and $\widetilde{L_{\mathrm{T}}}$ until completing the instruction or reaching the maximum timestep. The IFP can be optimized with an arbitrary RL algorithm using the samples collected from the environments. In our implementation, we use PPO [Schulman et al., 2017] for TALAR and all baselines. Note that during IFP training, the translator's parameters are fixed to prevent the translator from overfitting the current IFP.

5 Experiments

We conduct multiple experiments to evaluate TALAR and answer the following questions: (1) Can TALAR translate diverse NL instructions into a unique representation? (Section 5.1) (2) How does TALAR compare to traditional NLC-RL approaches in the instruction-following task? (Section 5.2) (3) Can TL serve as an abstraction for hierarchical RL? (Section 5.3) (4) How does every component influence the performance of TALAR? (Section 5.4)

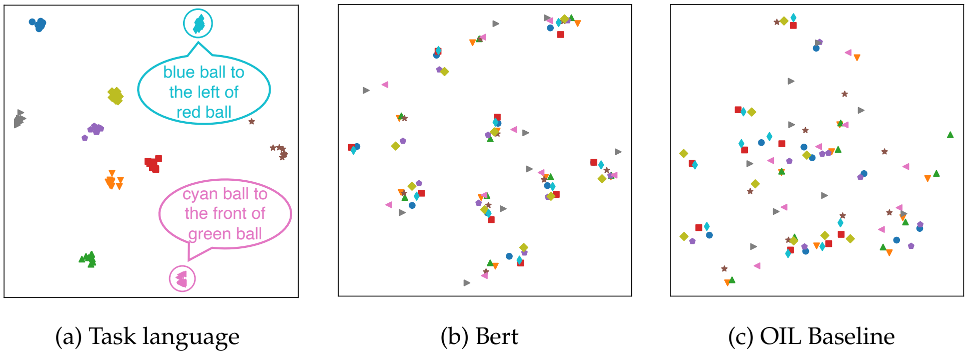
We perform experiments in CLEVR-Robot environment [Jiang et al., 2019], as shown in Figure 3. CLEVR-Robot is an environment for object interaction based on the MuJoCo physics engine [Todorov et al., 2012]. The environment contains five movable balls and an agent (silverpoint). In each trajectory, the agent aims to complete a human instruction in NL that represents moving a specific ball to one direction (i.e., one of `[front, behind, left, right]`) of a destination ball. For example, an NL instruction can be *Move the red ball to the left of the blue ball*, or *Can you push the yellow ball to the right of the green ball?* There are a total of 80 distinct human instructions. We use 18 different NL sentence patterns for each human instruction to describe it, yielding 1440 different NL instructions.

To acquire the task dataset, we first train a policy that could move any specified ball to a specified position with PPO algorithms. Then, this policy will collect 100,000 state transitions, each corresponding to one random ball movement. Then, each state transition is assigned an NL description. We use Bert-base-uncased [Devlin et al., 2019] as all pre-trained language models in our experiments. All experiments are performed with different random seeds five times, and the shaded area in the figures represents the standard deviation across all five trials. We refer readers to Appendix D for additional implementation details.



(a) Beginning of a task (b) Termination of a task

Figure 3: A visualization of CLEVR-Robot environment in our experiments. (a) In the beginning, one NL instruction is randomly sampled as *Can you move the cyan ball in front of the blue ball?* Then agent executes actions to complete the instruction. (b) The task terminates if achieving the goal or reaching the maximum timestep.



(a) Task language (b) Bert (c) OIL Baseline

Figure 4: The t-SNE representations of different types of NL encoding. Points with the same marker stand for the encoding of nine different NL expressions that describe the same human instruction. We add a slight noise to the overlapping points for better presentation. **(a)** The t-SNE representations of the TL output by the translator. **(b)** The encoding output by Bert model. **(c)** The encoding output by the language encoding layer of the OIL baseline (Bert-continuous in Section 5.2).

5.1 Task Language Development and Translation

We first verify whether the TALAR can translate diverse NL expressions into a unique representation. To answer the question, we randomly sample 10 different human instructions. Each instruction is expressed using nine NL sentence patterns for ninety NL sentences. Then, the optimized translator translates these NL sentences into TL. As depicted in Figure 4a, we project the resulting TL onto a two-dimensional plane using t-SNE [der Maaten et al., 2008]. Based on the projection results, we observe that TALAR learns a unique representation of NL. This conclusion can be obtained because TL can represent different NL expressions for the same human instruction in a close area. As a comparison, we also project the NL encoding directly output by a pre-trained Bert model, as shown in Figure 4b. The points produced by Bert are scattered everywhere on the plane, indicating that a pre-trained Bert model fails to represent diverse human instructions uniquely. Besides, as depicted in Figure 4c, an OIL baseline cannot distinguish the same human instruction with different NL expressions. This result suggests that the OIL baseline treats distinct NL expressions as distinct task objectives, which could slow policy learning. We refer readers to Appendix E.2 for more experiment results about the t-SNE projections.

Table 1: A summary of the final success rate (%) in instruction-following task with different sets of NL expressions. The results are averaged over 5 seeds, and each data is evaluated for 40 episodes.

Dataset Method	Training	Testing	Error-added
TALAR	**99.9± 0.1**	**78.3 ± 3.1**	**76.3 ± 3.6**
One-hot	86.5 ± 3.0	47.6 ± 2.5	62.1 ± 4.1
Bert-binary	64.0 ± 2.5	64.2 ± 5.0	67.8 ± 2.9
Bert-continuous	60.7 ± 1.9	54.0 ± 1.3	57.5 ± 2.1

In addition to the above results, we observe that the generated TL is interpretable to some extend. We use the TL generator to output the TL regarding all state transitions in the task dataset, and observe the resulting TL and its the corresponding NL descriptions. Consequently, the output of the predicate network is related to the destination ball. Figure 5 presents the frequency of five destination balls when the predicate network outputs 1. PredNet$_1$ and PredNet$_2$ clearly target the blue and purple balls, respectively. However, PredNet$_3$ is more difficult to interpret than the other two networks. There could be other relations other than with the destination ball.

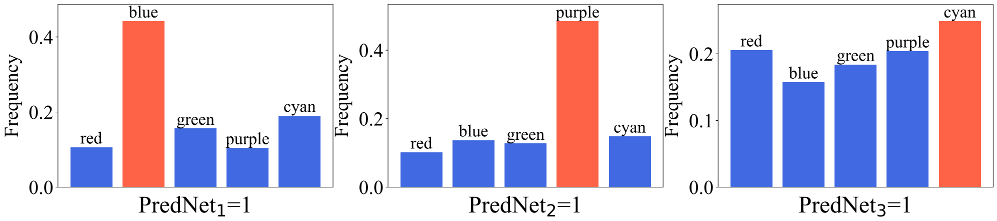


Figure 5: Frequency of five destination balls when a predicate network outputs a value of 1. Each bar stands for the frequency of the ball with a certain colour.

5.2 Performance of Instruction-Following Policy

.5With the optimized translator, we train an instruction-following policy in the CLEVR-Robot task, following the training process elaborated in Section 4.3. We first introduce three datasets of different NL expressions. (1) **Training set** contains nine NL sentence patterns (i.e., 720 NL instructions) for policy learning. All agents only interact with the training dataset when optimizing the policy. (2) **Testing set** contains nine NL sentence patterns (i.e., 720 NL instructions) that are different from the training set. (3) **Error-added set** contains the exact 720 NL instructions as the training set, with the addition of errors to each NL instruction, such as the word [the] being omitted. See Appendix D.1 for information regarding these three datasets and evaluation tasks.

Baselines for comparison. We consider multiple baselines that are built upon OIL architecture (i.e., standard NLC-RL): (1) **Bert-binary** processes the NL with a pre-trained Bert LM. The language encoding from the Bert is processed to a binary vector by a fully-connected network. This binary vector's size equals TL generated by TALAR. To ensure the differentiability, we use a reparameterization trick [Tokui and Sato, 2016] that converts continuous vector to binary vector. (2) **Bert-continuous** is similar to Bert-binary, except that it replaces the binary vector with a continuous vector of the same size. (3) **One-hot** encodes the representation of all possible NL instructions (including training, testing and error-added) to a three-dimensional tensor, where each instruction has its position.

Experimental results. Figure 6 presents the training curves on the instruction-following task with

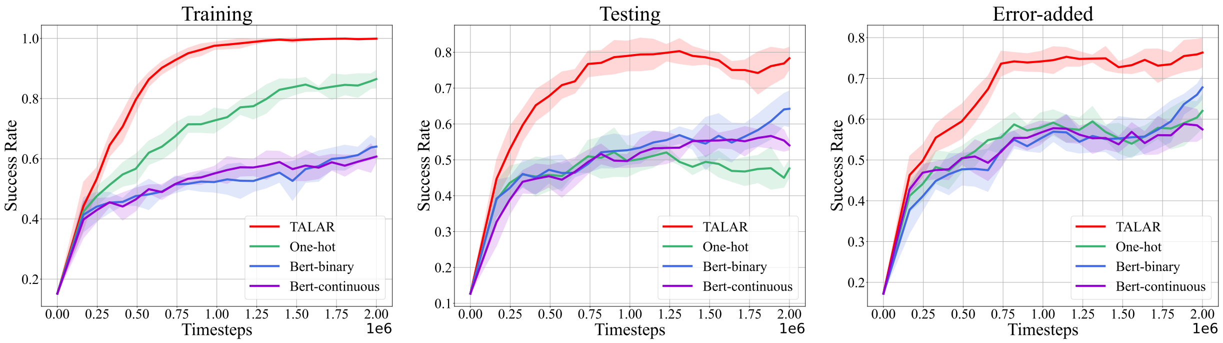


Figure 6: Training curves of different methods on three NL instruction datasets. The x-axis represents the total timesteps agent interacts with the environment, and the y-axis represents the success rate of completing instructions. The shaded area stands for the standard deviation over five random trials.

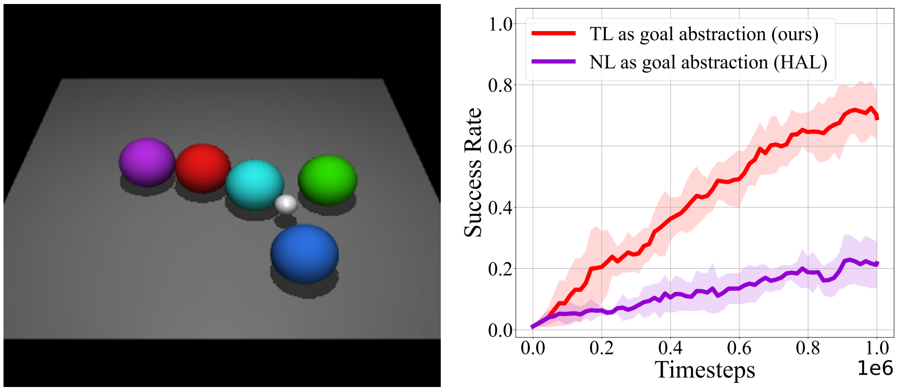
various NL instruction datasets, and Table 1 summarizes the final success rates of all methods. Overall, TALAR acquires a better instruction-following policy that increases the success rate by 13.4% relative to OIL baselines and adapts to previously unseen expressions of NL instruction. On the training NL instruction set, TALAR achieves a success rate greater than 99% within 2M timesteps, significantly faster than the two Bert-based baselines. Combined with the results in Section 5.1, the translator can effectively convert diverse NL expressions to a unique TL representation, which enables efficient policy learning. Besides, TALAR achieves a success rate greater than 76% in both testing and error-added sets, demonstrating greater capacity than baselines. While One-hot performs adequately on the training NL set, its generalization to the testing and error-added NL sets is limited. Two Bert-based baselines improve more slowly than TALAR on the training NL instruction set, which can be attributed to the fact that they simultaneously train a policy while acquiring skills and understanding NL. Bert's encoding of different NL expressions can be highly diverse, which adds complexity to OIL baselines to solve RL tasks; consequently, Bert-based baselines improve more slowly than TALAR during the training process.

5.3 TL as an Abstraction for Hierarchical RL

Previous experiments demonstrate that the resulting TL is a unique representation of the various NL expressions, which assists a policy in efficiently learning to follow NL instructions. In this section, we further explore the applicability of generated TL by examining if it can serve as an effective goal abstraction for hierarchical RL. Specifically, we train a high-level policy outputting a TL, instructing the IFP to complete a low-level task. We consider a baseline **HAL** [Jiang et al., 2019] for comparison. HAL takes advantage of the compositional structure of NL and directly makes decisions on the NL level. Following its original implementation, the high-level policy of HAL outputs the index of NL instruction. To ensure a fair comparison, HAL uses the IFP trained by TALAR as a low-level policy in our experiment. The high-level policies are trained with the PPO algorithm. We consider a long-term task based on the CLEVR-Robot environment, namely object arrangement, as shown in Figure 7a. The task objective of object arrangement is to arrange objects to satisfy all ten constraints that are implicitly contained in the task. Figure 7b presents the comparison results. The high-level policy that uses TL as a low-level goal abstraction performs significantly better than that using NL in terms of improving speed. This result shows that TL can be a helpful goal abstraction naturally compatible with hierarchical RL.

5.4 Ablation Study

We further conduct ablation experiments to verify how each component affects the performance of TALAR.

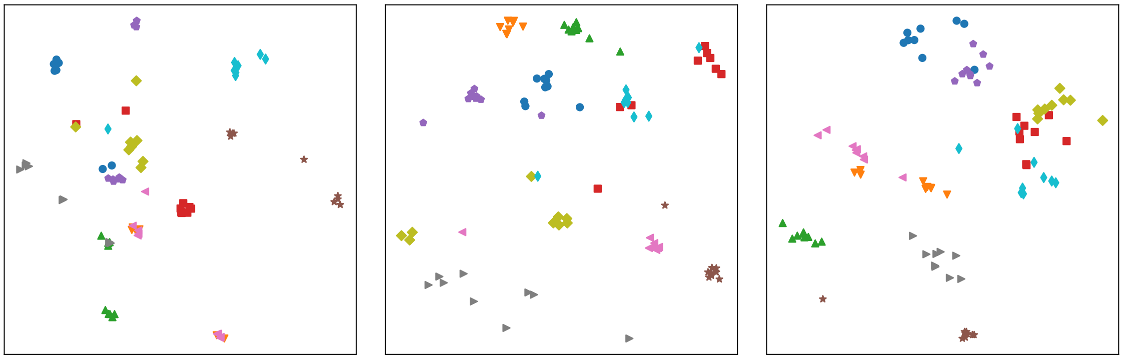


(a) An example of successful object arrangement.

(b) Success rate curves during the training process.

Figure 7: Experiments in a long-term object arrangement task. (a) A snapshot of successful object arrangement, which aims to arrange objects to satisfy all 10 human instructions at the same time. (b) Training curves of different goal abstractions in object arrangement.

Predicate representation. To evaluate the efficacy of predicate representation in TALAR, we replace it with binary/continuous vectors and derive two representations of TL, TL-binary and TL-continuous. In TL-binary/TL-continuous, a multi-layer perception network outputs the TL vector directly. Figure 8 shows the t-SNE projection of different kinds of TL representations. The results are computed based on the testing NL instruction set. For each human instruction with different NL expressions, the points produced by TL-predicate are more concentrated than those of TL-binary and TL-continuous. These results demonstrate the effectiveness of predicate representation for developing TL. Besides, Table 2 displays the final success rate of IFPs trained with various TL representations. Overall, all three types of representations are adequate for achieving a high task success rate on the training NL instruction set. However, predicate representation is more adaptable to unseen NL expressions and has a higher task success rate on testing/error-added NL instruction sets.



(a) TL-predicate

(b) TL-binary

(c) TL-continuous

Figure 8: The t-SNE projections of the task language in different kinds of representations, with the points generated from the testing NL instruction set.

VAE of the translator. TALAR employs a VAE for translator training. To demonstrate its efficacy, we introduce TALAR-MLP, which replaces the VAE in TALAR with a multi-layer perception (MLP) network and trains the translator using supervised learning loss. Then, we train an IFP using the MLP translator, and

Table 2: Comparisons of different representations of TL. The success rate (%) is averaged over 5 seeds.

Dataset TL rep.	Training	Testing	Error-added
Predicate	**99.9**	**78.3**	**76.3**
Binary	**99.8**	77.1	75.7
Continuous	**99.8**	**78.1**	66.3

the experiment results are depicted in Figure 9. Despite having comparable success rates on the training set, TALAR-VAE outperforms TALAR-MLP on the testing NL instruction set by 8.4% of success rate. This result suggests that VAE is advantageous when training a translator to generalize unseen NL expressions.

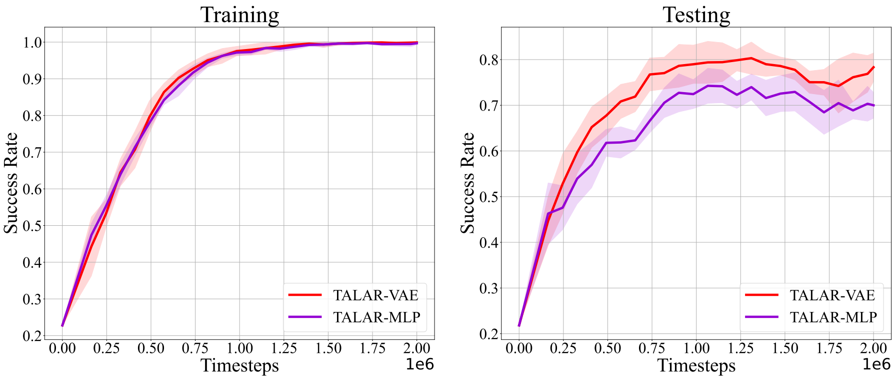
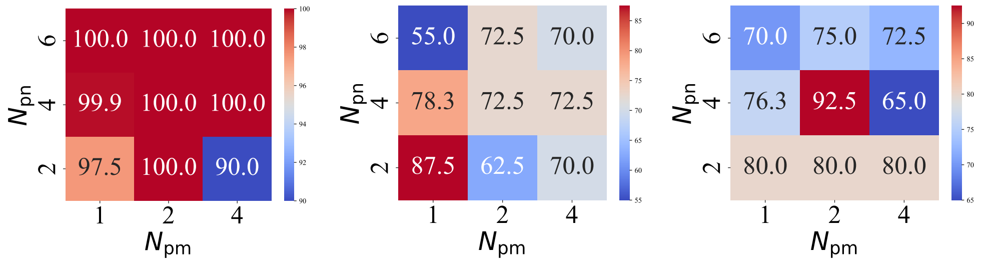


Figure 9: Training curves of IFP with different structures of the translator. See Appendix E.5 for the training curves on the error-added set.

Number of predicate modules and predicate networks. We conduct experiments to examine how the number of predicate modules/networks (i.e., N_{pm} and N_{pn}) in TALAR affects the performance. In our experiments, N_{pm} is selected from [1, 2, 4], while N_{pn} is selected from [2, 4, 6]. Figure 10 shows the experiment results. In general, greater N_{pm} and N_{pn} result in improved performance on the training NL instruction set (see Figure 10a). However, the experimental performance on the testing/error-added set is quite the opposite, as shown in Figure 10c when $N_{\text{pm}} = 4$ and $N_{\text{pn}} = 6$. This result could be attributed to the fact that, as the number of predicate modules and networks increases, the representation of TL becomes more complex (i.e., the vector size increases), making it more difficult for the policy to follow the TL. Besides, we also observe that, within a specific range of values ($N_{\text{pm}} \leq 2$ and $N_{\text{pn}} \leq 4$), larger N_{pn} and N_{pm} would bring a better performance. These results serve as a guide for selecting appropriate hyper-parameters. Due to space constraints, experiments on TALAR involving a variety of argument networks are presented in Appendix E.4.

6 Conclusion

This paper focuses on the topic of NLC-RL. We suggest that NL is an unbounded representation of human instruction, thereby imposing a substantial additional burden on the policy when solving RL tasks. To alleviate the burden, we investigate a new IOL scheme for NLC-RL by developing TL, which is task-related, and a unique representation of human instruction. Through our experiments, we verify that the resulting TL



(a) Training

(b) Testing

(c) Error-added

Figure 10: Ablation study on the number of predicate modules/networks. The values in the heat map represent the success rate of IFPs trained for 2M timesteps, with different parameter configurations of N_{pm} and N_{pn}.

can uniquely represent human instructions with diverse NL expressions and is interpretable to some extent. Besides, the policy following TL can quickly learn to complete the instructions with a high success rate and adapts to previously unseen NL expressions. Moreover, the resulting TL is an effective goal abstraction of a low-level policy that serves as the basis for hierarchical RL.

Although TALAR can effectively train a competent instruction-following policy, there are limitations. TALAR develops the task language using a static task dataset and, therefore, can not be directly applied to an open environment task. It is possible to mitigate this issue by dynamically extending the task dataset and fine-tuning the TL generator/translator during the policy learning process. Besides, TALAR requires a manual reward function for policy training, which may be inaccessible if the reward design is complex. Fortunately, there have been well-validated methods for solving sparse reward problems [Andrychowicz et al., 2017, Nair et al., 2018, Riedmiller et al., 2018], which is an effective substitute for the manual reward function. Finally, it would be interesting to involve the basic properties of predicate relationships (such as transitivity, reflexivity, and symmetry) when training the TL generator, which makes the resulting TL more meaningful and self-contained. We hope future research will investigate these intriguing questions and make strides toward training agents that interact with humans more effectively.

References

R. Agarwal, C. Liang, D. Schuurmans, and M. Norouzi. Learning to generalize from sparse and underspecified rewards. In *ICML*, 2019.

A. Akakzia, C. Colas, P. Oudeyer, M. Chetouani, and O. Sigaud. Grounding language to autonomously-acquired skills via goal generation. In *ICLR*, 2021.

M. Andrychowicz, D. Crow, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, P. Abbeel, and W. Zaremba. Hindsight experience replay. In *NeurIPS*, 2017.

M. Asai. Unsupervised grounding of plannable first-order logic representation from images. In *ICAPS*, 2019.

D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

A. G. Barto and S. Mahadevan. Recent advances in hierarchical reinforcement learning. *Discret. Event Dyn. Syst.*, 13:41–77, 2003.

M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. The arcade learning environment: An evaluation platform for general agents. *J. Artif. Intell. Res.*, 47:253–279, 2013.

V. Blukis, C. Paxton, D. Fox, A. Garg, and Y. Artzi. A persistent spatial semantic representation for high-level natural language instruction execution. In *CoRL*, 2022.

A. Borgida. On the relative expressiveness of description logics and predicate logics. *Artif. Intell.*, 82:353–367, 1996.

D. S. Chaplot, K. M. Sathyendra, R. K. Pasumarthi, D. Rajagopal, and R. Salakhutdinov. Gated-attention architectures for task-oriented language grounding. In *AAAI*, 2018.

K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *EMNLP*, 2014.

G. Cideron, M. Seurin, F. Strub, and O. Pietquin. Higher: Improving instruction following with hindsight generation for experience replay. In *SSCI*, 2020.

V. der Maaten, Laurens, and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9: 2579–2605, 2008.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*. Association for Computational Linguistics, 2019.

T. Eccles, Y. Bachrach, G. Lever, A. Lazaridou, and T. Graepel. Biases for emergent communication in multi-agent reinforcement learning. In *NeurIPS*, 2019.

F. Hill, S. Mokrá, N. Wong, and T. Harley. Human instruction-following with deep reinforcement learning via transfer-learning from text, 2021.

E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *ICLR*, 2017.

Y. Jiang, S. Gu, K. Murphy, and C. Finn. Language as an abstraction for hierarchical deep reinforcement learning. In *NeurIPS*, 2019.

N. K. Jong, T. Hester, and P. Stone. The utility of temporal abstraction in reinforcement learning. In *AAMAS*, 2008.

Y. Kang, T. Wang, and G. de Melo. Incorporating pragmatic reasoning communication into emergent language. In *NeurIPS*, 2020.

D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR*, 2014.

R. Kozierok, J. S. Aberdeen, C. Clark, C. D. Garay, B. Goodman, T. Korves, L. Hirschman, P. L. McDermott, and M. W. Peterson. Hallmarks of human-machine collaboration: A framework for assessment in the DARPA communicating with computers program, 2021.

A. Lazaridou and M. Baroni. Emergent multi-agent communication in the deep learning era, 2020.

A. Lazaridou, A. Peysakhovich, and M. Baroni. Multi-agent cooperation and the emergence of (natural) language. In *ICLR*, 2017.

S. Li, Y. Zhou, R. E. Allen, and M. J. Kochenderfer. Learning emergent discrete message communication for cooperative reinforcement learning. In *ICRA*, 2022.

J. Luketina, N. Nardelli, G. Farquhar, J. N. Foerster, J. Andreas, E. Grefenstette, S. Whiteson, and T. Rocktäschel. A survey of reinforcement learning informed by natural language. In *IJCAI*, 2019.

T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *ICLR*, 2013.

D. K. Misra, A. Bennett, V. Blukis, E. Niklasson, M. Shatkhin, and Y. Artzi. Mapping instructions to actions in 3d environments with visual goal prediction. In *EMNLP*, 2018.

O. Nachum, S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. In *NeurIPS*, 2018.

O. Nachum, H. Tang, X. Lu, S. Gu, H. Lee, and S. Levine. Why does hierarchy (sometimes) work so well in reinforcement learning?, 2019.

A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *ICRA*, 2018.

F. A. Oliehoek and C. Amato. *A Concise Introduction to Decentralized POMDPs*. Springer Briefs in Intelligent Systems. Springer, 2016. ISBN 978-3-319-28927-4.

A. Pagnoni, K. Liu, and S. Li. Conditional variational autoencoder for neural machine translation, 2018.

S. Patel, S. Wani, U. Jain, A. G. Schwing, S. Lazebnik, M. Savva, and A. X. Chang. Interpretation of emergent communication in heterogeneous collaborative embodied agents. In *ICCV*, 2021.

S. Pateria, B. Subagdja, A. Tan, and C. Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Comput. Surv.*, 54(5):109:1–109:35, 2021.

K. Pertsch, Y. Lee, and J. J. Lim. Accelerating reinforcement learning with learned skill priors. In *CoRL*, 2020.

D. Pomerleau. Efficient training of artificial neural networks for autonomous navigation. *Neural computation*, 3:88–97, 1991.

M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley Series in Probability and Statistics. Wiley, 1994.

A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *J. Mach. Learn. Res.*, 22:268:1–268:8, 2021.

M. A. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. V. de Wiele, V. Mnih, N. Heess, and J. T. Springenberg. Learning by playing solving sparse reward tasks from scratch. In *ICML*, 2018.

I. Rivera-Trigueros. Machine translation systems and quality assessment: a systematic review. *Lang. Resour. Evaluation*, 56:593–619, 2022.

S. J. Russell and P. Norvig. *Artificial intelligence - a modern approach: the intelligent agent book*. Prentice Hall series in artificial intelligence. Prentice Hall, 1995. ISBN 978-0-13-103805-9.

J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

D. Simões, N. Lau, and L. P. Reis. Multi-agent deep reinforcement learning with emergent communication. In *IJCNN*, 2019.

F. Stahlberg. Neural machine translation: A review. *J. Artif. Intell. Res.*, 69:343–418, 2020.

R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. *IEEE Trans. Neural Networks*, 9(5): 1054–1054, 1998.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *IROS*, 2012.

S. Tokui and I. Sato. Reparameterization trick for discrete variables, 2016.

A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. Feudal networks for hierarchical reinforcement learning. In *ICML*, 2017.

X. Yang, Z. Ji, J. Wu, Y. Lai, C. Wei, G. Liu, and R. Setchi. Hierarchical reinforcement learning with universal policies for multistep robotic manipulation. *IEEE Trans. Neural Networks Learn. Syst.*, 33:4727–4741, 2022.

APPENDIX

A Discussion

This section will present examples to illustrate the predicate representation better.

A.1 What Is the Predicate Representation?

Overall, predicate representation utilizes the form of predicate expressions, which can model various relationships in the tasks. Specifically, in this paper, we utilize a discrete binary vector to represent the truth value of multiple anonymous predicates and the arguments of these predicates. For example, in a predicate representation vector, [1, 0, 1, 0, 1, 0, 0], the first code [1] stands for the value of an anonymous predicate is True, the red and blue codes stand for the indexes of two arguments of this predicate, respectively.

A.2 What Are the Advantages of the Predicate Representation?

In the main body (Section 4), we have mentioned that predicate representation is expressive. We suggest that such expressiveness comes from two key advantages of the predicate representation: *compositional structure* and *interpretability*. We first discuss the **compositionality**, which refers to the ability of a language to denote novel composite meanings. For example, if a language can represent *blue circle* and *red square*, it can represent *blue square* as well. This language has a compositional structure. The compositionality is seen both as a fundamental feature of natural language and as a pre-condition for a language to generalize at scale [Lazaridou and Baroni, 2020]. Predicate representation naturally has compositional structure due to the fact that it can denote the
composite meanings by changing the truth value of predicates.

Next, we talk about the **interpretability**. Predicate representation uses multiple predicate expressions, such as `Pred(a1,a2)`, to describe the relationships in a specific environment. Previous works have found that, even if the predicate symbols are machine-generated by a learning system and anonymous to humans, they still offer *some extent of interpretability*. For example, if a learning system generates a predicate expression `Pred(cat,grassland)=True` for Figure 11, we can guess that `Pred` means [stand] or [above]. When more figures and predicate expressions are provided, the actual meaning of these anonymous predicate symbols would be more apparent to humans.

In conclusion, we implement the IOL architecture utilizing predicate representation because nearly all tasks contain relationships. For instance, an Atari game [Bellemare et al., 2013] contains a relationship between the agent and the antagonist, a first-person shooter game contains a relationship between the gun and the bullets, a stock trading task contains a relationship between stocks and price, etc. Even if some tasks are extremely abstract and may have no relationships, the predicate representation can be replaced with TL in binary or continuous representations, which have demonstrated their ability to develop a task language in our experiments (Section 5.4).

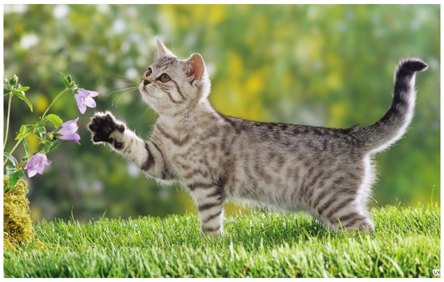


Figure 11: An illustration of the interpretability of predicate representation. For an anonymous predicate expression: `Pred(cat,grassland)=True`, we can guess that `Pred` represents [above].

B Additional Related Work

Hierarchical Reinforcement Learning (HRL) approaches are promising tools for solving complex decision-making tasks [Pateria et al., 2021]. HRL solves the problem by decomposing it into simpler sub-tasks using a hierarchy of policies learned by RL [Vezhnevets et al., 2017, Jong et al., 2008, Nachum et al., 2019]. For example, to put a drink into a fridge, you will (1) take up the drink, (2) open the fridge door, (3) put down the drink, and (4) close the fridge door. Putting a drink into a fridge is a long-term task, while these four are sub-tasks. Typically, a high-level policy is trained to decompose the main task into sub-tasks (three steps in the example), and low-level policies (or single policy) are trained to complete these sub-tasks. A *core problem* in HRL is how to represent the goal of the sub-task (i.e., goal representation) that the low-level policy follows. Previous works have investigated both concrete and abstract goal representations of a sub-task. Concrete goal representation can be the *target position* to reach [Yang et al., 2022], or a *target state* [Nachum et al., 2018]. As for abstract goal representation, it can directly be natural language [Jiang et al., 2019] or an encoding of action primitives [Pertsch et al., 2020]. Back to the topic of this paper, language is a natural goal representation for the sub-tasks, such as *open the fridge door* in the above example. Through our experiments (Section 5.3), we observe that the resulting TL is an effective abstraction of the goal of the sub-task, which demonstrates the applicability of TL to work together with HRL.

C Algorithm Descriptions

Algorithm 1, 2, and 3 present the training procedures of the TL generator, the translator and the instruction-following policy, respectively.

Algorithm 1 Training procedure of the TL generator.

```
Input: task dataset $\mathcal{D} = \{(s, s', L_\mathrm{N})_i\}$, pre-trained Bert model $b$.
Output: the optimized TL generator.
Initialize the TL generator $g_\theta$.
while training not complete do
    Sample a batch of data from $\mathcal{D}$.
    Update $\theta$ to minimize the MLM loss (Eq.(2)).
end while
return the optimized TL generator $g_\theta$.
```

Algorithm 2 Training procedure of the translator.

```
Input: task dataset $\mathcal{D} = \{(s, s', L_\mathrm{N})_i\}$, the optimized TL generator $g_\theta$, and the pre-trained Bert model.
Output: the optimized TL generator.
Initialize the translator $t_{\phi_1,\phi_2}$ with parameters $\phi_1$ and $\phi_2$.
while training not complete do
    Sample a batch of data $\{(s, s', L_\mathrm{N})_j\}$ from $\mathcal{D}$.
    // Compute the target task language which VAE aims to recover.
    Calculate the task language $L_\mathrm{T} = g_\theta(s, s')$.
    Update $\phi_1, \phi_2$ to minimize the VAE loss (Eq.(3)).
end while
return the optimized translator $t_{\phi_1,\phi_2}$.
```

D Implementation Details

In our experiments, we utilize the open-sourced RL repository, stable-baselines3 [Raffin et al., 2021], to implement the RL training. All experiments are run for five times with different random seeds. We will next introduce the tasks for evaluation and the hyper-parameters used in our experiments.

Algorithm 3 Training procedure of the instruction-following policy.

```
Input: the optimized translator $t_{\phi_1,\phi_2}$.
Output: the optimized instruction-following policy.
Initialize the policy function $\pi$, and the value function.
while training not complete do
    Sample a NL instruction $L_{\mathrm{N}}$ from the environment.
    Generate corresponding task language $\widetilde{L_{\mathrm{T}}} = t_{\phi_1,\phi_2}(L_{\mathrm{N}})$.
    // Collecting samples
    while episode not terminal do
        Observe current state $s_t$.
        Execute action $a_t \sim \pi(\cdot|s_t, \widetilde{L_{\mathrm{T}}})$, and receive a reward $r_t$ from the environment.
    end while
    // Training
    Update the policy and value functions based on the samples collected from the environment.
end while
return the optimized policy $\pi$.
```

D.1 Tasks for Evaluation

Instruction following task. As depicted in Figure 3, our experiments are conducted in a CLEVR-Robot environment where an agent manipulates five balls to complete human instructions. The observation space is $s \in \mathbb{R}^{10}$, which represents the location of each object, and $|A| = 40$ corresponds to selecting and pushing an object in one of the eight cardinal directions. At each timestep, the reward is equal to (distance to the target position at the previous step) minus (distance to the target position at the current step), with +5 for success and -5 for failure. Some previous OIL-based methods attempt to train the IFP in a sparse reward environment [Agarwal et al., 2019, Cideron et al., 2020]. We relax the sparse reward constraint to dense reward for two reasons: (1) We are more concerned with exploring a new learning scheme for NLC-RL than with resolving the sparse reward problem; (2) Most NLC-RL approaches address the sparse reward problem using HER [Andrychowicz et al., 2017], which relabels the origin goal in the trajectory with the actual achieved state. Nonetheless, the relabeling procedure is contingent on the transformation from the state to the goal, which necessitates additional human annotation or program design.

We use eighteen different NL sentence patterns to express each human instruction. Take a human instruction, blue ball to the right of the green ball, for example, its corresponding NL instructions (i.e., or eighteen NL sentence patterns) can be one of the:

- —(**Training set**)—
- Push the blue ball to the right of the green ball.
- Can you push the red ball to the right of the green ball?
- Can you help me push the red ball to the right of the green ball?
- Is the red ball right of the green ball?
- Is there any red ball right the green ball?
- The red ball moves to the right of the green ball.
- The red ball is being pushed to the right of the green ball.
- The red ball is pushed to the right of the green ball.
- The red ball was moved to the right of the green ball.
- —(**Testing set**)—
- Keep the red ball right of the green ball.
- Move the red ball to the right of the green ball.
- Can you move the red ball to the right of the green ball?

- Can you keep the red ball to the right of the green ball?
- Can you help me move the red ball to the right of the green ball?
- Can you help me keep the red ball to the right of the green ball?
- The red ball is being moved to the right of the green ball.
- The red ball is moved to the right of the green ball.
- The red ball was pushed to the right of the green ball.

We also consider a set of errors in the NL sentence:

- Missing a [the].
- Incorrect use of prepositions, e.g., using [*on front of*] to replace [*in front of*].
- Incorrect use of phrase, including *in behind of*, *in left of*, and *in right of*.
- Oversimplifying the expression, e.g., move red ball right green ball.

In our experiments, there are three kinds of datasets: training, testing, and error-added. In the training set, human instructions are expressed using nine NL sentence patterns, while the remaining nine NL sentence patterns are used in the testing set. The error-added set utilizes the same NL nine sentence patterns as the training set, but each sentence has at least one of the errors listed above. At the start of each trajectory, the environment randomly samples a human instruction and an NL sentence pattern to express the human instruction. By constructing these three datasets, we simulate the scenario in an open environment where different individuals instruct the robots using their linguistic preferences.

High-level object arrangement task. Object arrangement task is proposed by [Jiang et al., 2019], which aims to rearrange the objects in the environment to satisfy all ten implicit constraints. At the start of a trajectory, the environment resets the position of all balls to a random location. At each time step, the reward agent receives a reward equal to (number of constraints satisfied at current timestep) - (number of constraints satisfied at previous timestep). Following [Jiang et al., 2019], the precise arrangement constraints are: (1) red ball to the right of purple ball; (2) green ball to the right of red ball; (3) green ball to the right of cyan ball; (4) purple ball to the left of cyan ball; (5) cyan ball to the right of purple ball; (6) red ball in front of blue ball; (7) red ball to the left of green ball; (8) green ball in front of blue ball; (9) purple ball to the left of cyan ball; (10) blue ball behind the red ball.

D.2 Hyper-Parameters

The hyper-parameters for implementing TALAR are presented in Table 3.

Table 3: Hyper-parameters in our experiments.

Hyper-parameters	**Value**
N_{a}	2
N_{pn}	4
N_{pm}	1
Size of ArgNet's output	5
Learning rate (LR) for $\mathcal{L}_{\mathrm{MLM}}$	3e-4
LR for $\mathcal{L}_{\mathrm{VAE}}$	3e-4
VAE encoder network	[256, 256, 32], relu
VAE decoder network	[256, 256, $\lvert L_{\mathrm{T}} \rvert$], relu
Predicate network	[128, 128, 2], relu
Argument network	[128, 128, 5], relu
PPO epoch	10
PPO policy LR	3e-4
PPO value LR	3e-4
PPO policy network	[32, 64, 64], tanh
PPO value network	[32, 64, 64], tanh
PPO mini-batch size	128
PPO nums of mini-batch	160

E Additional Experiment Results

E.1 Training IFP with Different Number of NL Sentence Patterns

We evaluate the robustness of different methods by training an IFP on the tasks with 1, 5, and 9 NL sentence patterns for 2M timesteps. Table 4 presents the experiment results. As the number of sentence patterns increases, TALAR nearly maintains a near 100% success rate. Since TALAR could uniquely represent the NL expressions in different sentence patterns, the IFP policy that follows TL can comprehend the instructions more easily and learn to manage the task rapidly. This result indicates that learning a unique representation of human instruction benefits policy learning. In contrast, as the number of NL sentence patterns increases, the performance of baselines declines significantly because they must simultaneously comprehend more NL expressions and learn the skills to manage the task.

Table 4: A summary of the final success rate (%) on the training set with a different number of NL sentence patterns. Each IFP is trained for 2M timesteps and evaluated for 40 episodes. The results are averaged over 5 seeds.

Pattern nums. Method	1	5	9
TALAR	**100%**	**100%**	**99.9%**
One-hot	98.2%	91.8%	86.5%
Bert-binary	82.0%	60.9%	64.0%
Bert-continuous	95.3%	63.1%	60.7%

E.2 Complete Results of T-SNE Projection of Different Representations

Figure 12 presents the total experimental results of the t-SNE projection of various representations on three NL instruction datasets. **TL-predicate, TL-continuous and TL-binary** stand for TL with predicate, continuous, and binary representations. **Bert-encoding** is the output of the Bert model. **Bert-continuous** is the output of the OIL baseline's NL feature layer. We observe that, on the training NL instruction set, the t-SNE projections of the three TL representations are considerably more agminated than those of Bert encoding and Bert-continuous, indicating that IOL is an effective method for learning a unique representation. However, in the testing and error-added sets, the TL-predicate representation performs significantly better than TL-binary and TL-continuous, as its points with same marker are more concentrated. This result demonstrates that predicate representation is more resistant to unseen NL expressions and produces a more concentrated representation. In contrast, the t-SNE projections of Bert encoding and Bert-continuous are dispersed throughout the plane, inducing NL understanding burden for the policy learning.

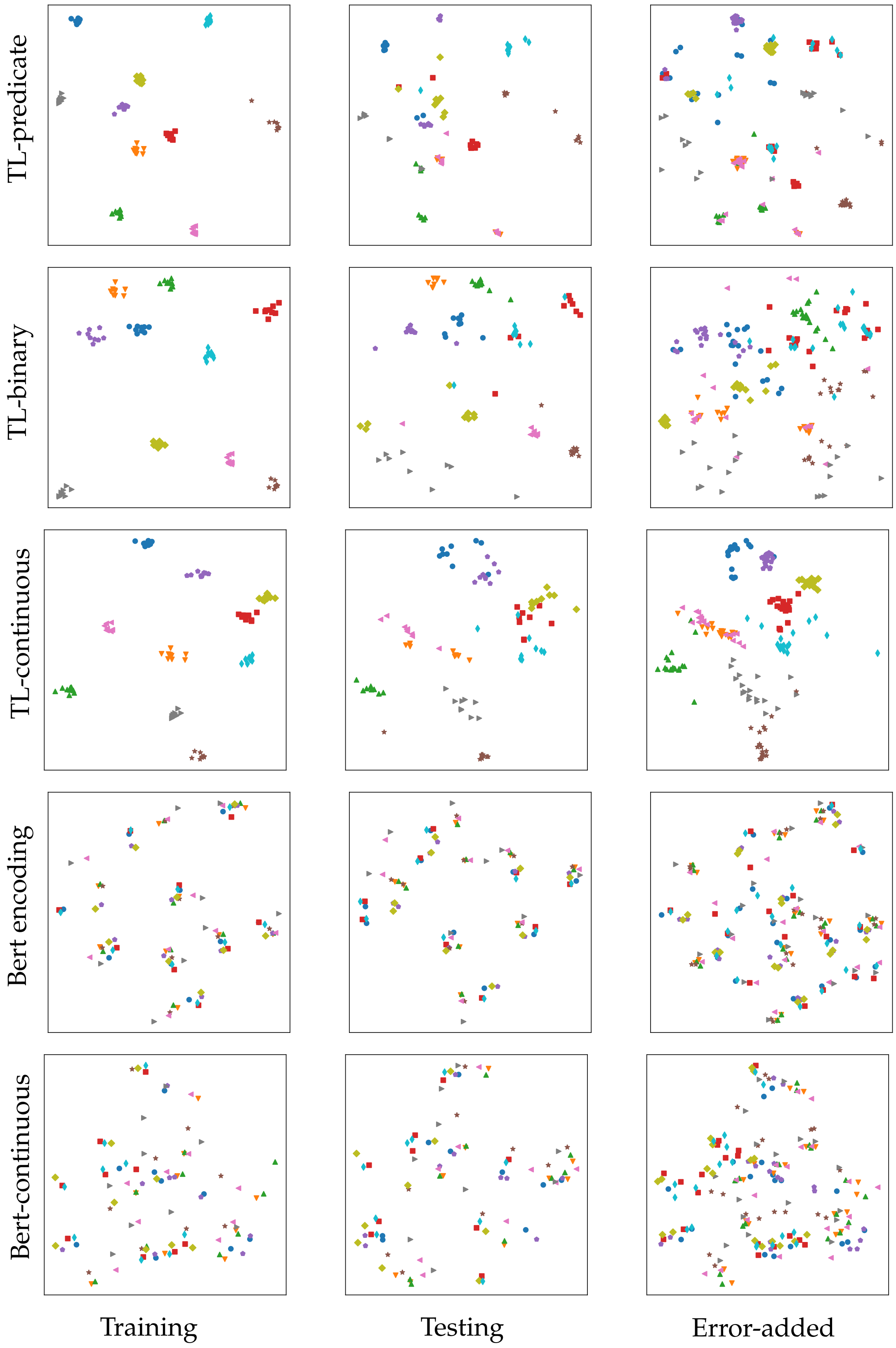
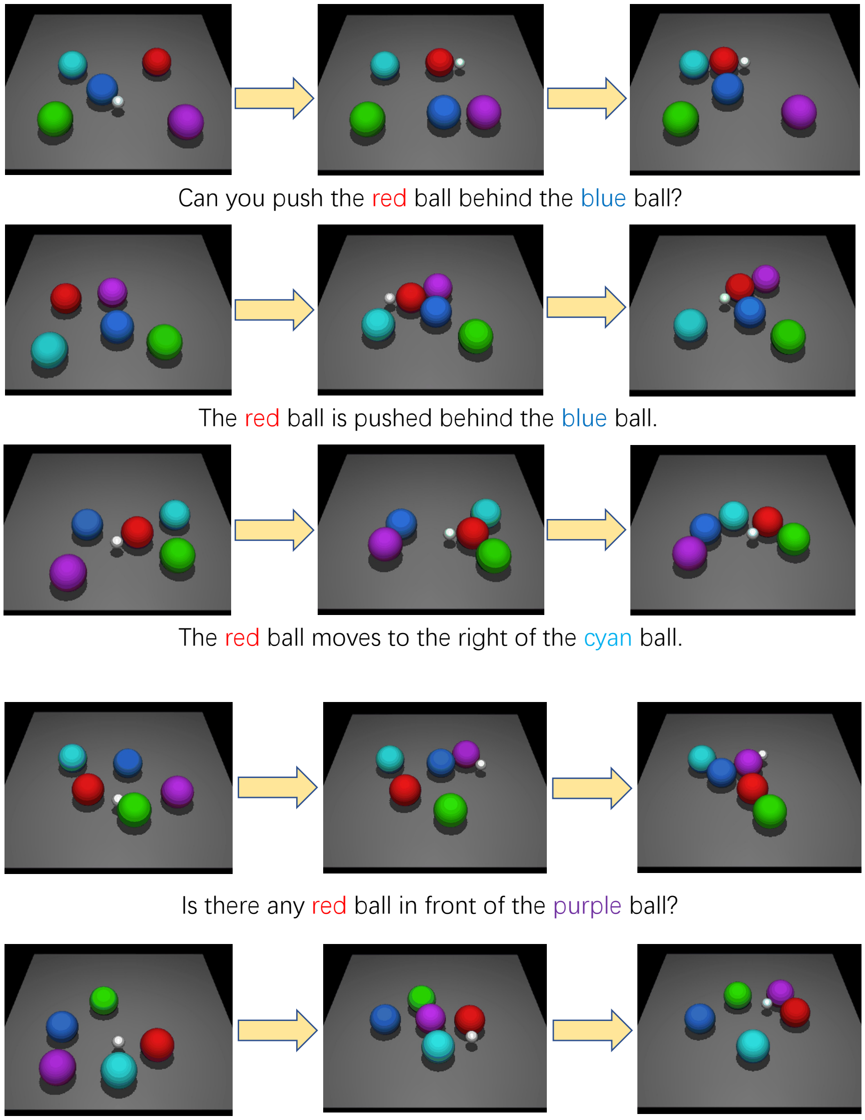


Figure 12: The t-SNE projection of different representations on three NL expressions datasets. Each row represents one kind of representation, and each column stands for one kind of NL expression. Points with the same marker encode nine different NL expressions that describe the same human instruction.

E.3 Deployment Examples of IFP Trained by TALAR

We visualize the performance of the IFP trained by TALAR in the CLEVR-Robot environment. Figure 13 presents the policy performance following different NL instructions. IFP can rapidly complete the tasks by moving related balls for NL instructions with different NL expressions.



Can you push the red ball behind the blue ball?

The red ball is pushed behind the blue ball.

The red ball moves to the right of the cyan ball.

Is there any red ball in front of the purple ball?

The red ball was moved in front of the purple ball.

Figure 13: A visualization of the TALAR's IFP deployment process.

E.4 Performance of TALAR With Different Number of the Argument Networks

Figure 14 shows the training curves of TALAR with different number of argument networks, when $N_{\text{pm}} = 1$ and $N_{\text{pn}} = 4$. The experiment results indicate that TALAR is not sensitive to N_{a}, and TALAR with $N_{\text{a}} = 2$ performs slightly better than the other two parameters.

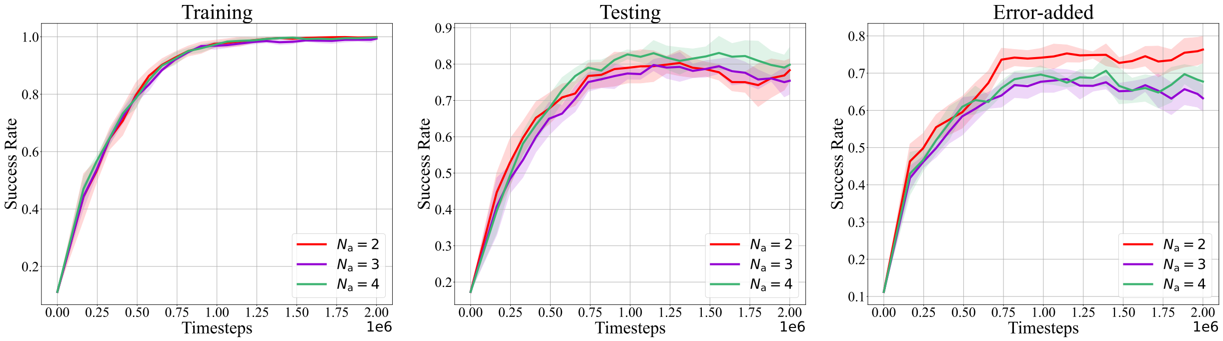


Figure 14: Training curves of TALAR with different number of argument networks. The x-axis represents the total timesteps agent interacts with the environment, and the y-axis represents the success rate of completing instructions. The shaded area stands for the standard deviation over five random trials.

E.5 Training Curves of TALAR-MLP

Figure 15 shows the ablation study on the VAE used in TALAR, where TALAR-MLP replaces the VAE in TALAR with an MLP network and trains the translator with a supervised learning loss. The experimental results indicate that TALAR-VAE is more robust on unseen NL instructions than TALAR-MLP, while they achieve similar performance on the training dataset.

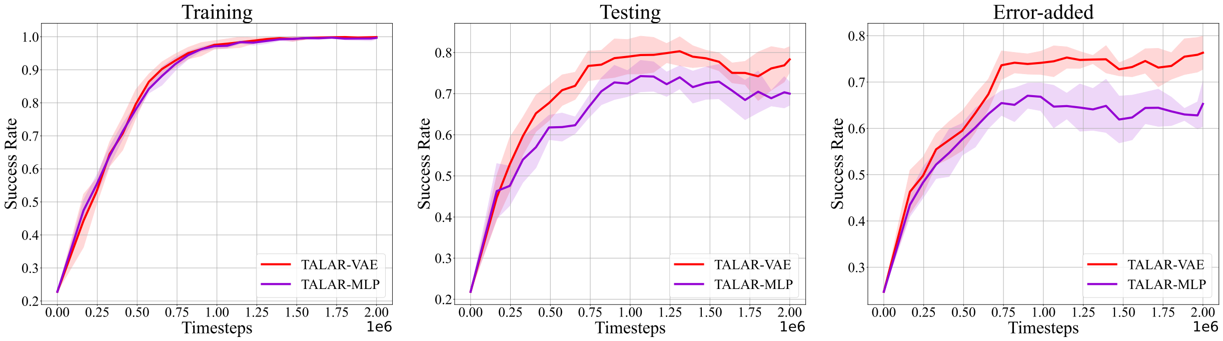


Figure 15: Ablation study on the VAE used in TALAR. The x-axis represents the total timesteps agent interacts with the environment, and the y-axis represents the success rate of completing instructions. The shaded area stands for the standard deviation over five random trials.

F Retrievals for Notations and Abbreviations

Table 5: Notations and abbreviations in this paper.

Name	Meaning
Notations	
L_{T}	task language
L_{N}	natural language
q_{ϕ_1}	encoder with parameters ϕ_1
p_{ϕ_2}	decoder with parameters ϕ_2
g_θ	TL generator with parameters θ
Abbreviations	
NL	Natural Language
TL	Task Language
LM	Language Model
RL	Reinforcement Learning
PM	Predicate Module
IOL	Inside-Out Learning
OIL	Outside-In Learning
VAE	Variational Auto-Encoder
MLM	Masked Language Modelling
NLC-RL	Natural Language-Conditioned Reinforcement Learning