InCLET: Large Language Model In-context Learning can Improve Embodied Instruction-following

A Preprint

Peng-Yuan Wang[1,2,3,*], Jing-Cheng Pang[1,2,3,*], Chen-Yang Wang[2,*], Xu-Hui-Liu[1,2,3], Tian-Shuo Liu[1,2,3], Si-Hang Yang[1,2,3], Hong Qian[4], Yang Yu[1,2,3,◇]

[1] National Key Laboratory for Novel Software Technology, Nanjing University, China
[2] School of Artificial Intelligence, Nanjing University, China
[3]Polixir.ai
[4] School of Computer Science and Technology, East China Normal University
[*] Equal contribution
[◇] Corresponding: yuy@nju.edu.cn

Abstract

Natural language-conditioned reinforcement learning (NLC-RL) empowers embodied agent to complete various tasks following human instruction. However, the unbounded natural language examples still introduce much complexity for the agent that solves concrete RL tasks, which can distract policy learning from completing the task. Consequently, extracting effective task representation from human instruction emerges as the critical component of NLC-RL. While previous methods have attempted to address this issue by learning task-related representation using large language models (LLMs), they highly rely on pre-collected task data and require extra training procedure. In this study, we uncover the inherent capability of LLMs to generate task representations and present a novel method, in-context learning embedding as task representation (MethodName). MethodName is grounded on a foundational finding that LLM in-context learning using trajectories can greatly help represent tasks. We thus firstly employ LLM to imagine task trajectories following the natural language instruction, then use in-context learning of LLM to generate task representations, and finally aggregate and project into a compact low-dimensional task representation. This representation is then used to train a human instruction-following agent. We conduct experiments on various embodied control environments and results show that MethodName creates effective task representations. Furthermore, this representation can significantly improve the RL training efficiency, compared to the baseline methods.

1 Introduction

Developing robots capable of executing tasks following human instructions is a highly attractive area of research (Brohan et al., 2022; Pang et al., 2023; Jiang et al., 2020; Brohan et al., 2023). Natural language-conditioned reinforcement learning (NLC-RL) has emerged as a powerful approach in this field, as it focuses on training agents to perform tasks specified by natural language instructions (Pang et al., 2024c,b; Luketina et al., 2019). The key of NLC-RL lies in processing the complex and diverse natural language (NL) into a *task representation*, which is then exposed to agent to complete the control task.

Traditional methods typically employ pre-trained large language models (LLMs) such as BERT (Kenton and Toutanova, 2019) and Llama (Dubey et al., 2024) to extract semantic information directly from natural language. They convert the NL instruction into LLM embeddings as the task representations, and then train a policy conditioned on these embeddings. However, these embeddings generated by LLMs are typically learned independently of the RL task, making it challenging to capture the task-related information in the natural language instruction. An alternative method is to train a natural language translator that converts natural language instructions into a low-dimensional machine-readable representations, leveraging pre-collected data from environmental interactions (Pang et al., 2024c; Volovikova et al., 2024). Despite their potential, such methods are heavily dependent on high-quality interaction data and necessitate a separate training process for translator, which in turn increases the cost and reduces the flexibility of the approach. This highlights the pressing need to *develop efficient task representations that require minimal additional efforts*.

In this work, we introduce a novel method, in-context learning embedding as task representation (MethodName), which produces effective task representation for NLC-RL without relying on pre-collecting environment data or additional training. Instead of directly utilizing LLM's embedding or additionally training a translator, MethodName leverages the in-context learning paradigm to extract task representations, capturing more intrinsic information, which is inspired by (Hendel et al., 2023). MethodName consists of three components: (1) an in-context generator that generates few task-related trajectories; (2) a task representation extractor that extracts task representation from generated few trajectories; (3) a policy that solves concrete RL tasks given task representation from human instructions. Specifically, in the in-context generator, we leverage the extensive knowledge of LLMs to automatically generate (initial state, terminal state) trajectory pairs about task, which are used for task representation extraction. In the second module, we format these as *"[Initial state]→[Terminal state]"*, extracting the hidden state at the '→' position. These hidden states are then fused to form the task representation. In the third module, we combine the state and task representations for policy training in reinforcement learning. To address the high dimensionality of the task representation space, we employ random projection to map it onto a much lower-dimensional subspace, making the representation more efficient for training. In contrast to previous methods, MethodName obtains effective task representations without requiring any task-specific information or additional training.

We justify the effectiveness of InCLET method through theoretical analysis, which shows that our method achieves tighter error bound compared to traditional methods. Besides, through extensive experiments in two embodied control environments: FrankaKitchen (Gupta et al., 2020) and CLEVR-Robot (Jiang et al., 2019), we demonstrate that the policy learned by MethodName outperforms previous methods, in terms of the ability to follow different natural language instructions and adapt to previously unseen NL instructions. Furthermore, we verify the source of MethodName's effectiveness, by performing t-SNE (Van der Maaten and Hinton, 2008) dimensionality reduction on the task representations for visualization. We observe that MethodName effectively separates the representations of different tasks, indicating the feasibility of the way to leverage in-context learning to extract task representations. Finally, we conducted an ablation study to analyze the impact of each module in MethodName on the overall performance.

We highlight the main contributions of our work as follows:

- We successfully verify that the in-context learning capabilities of LLMs can be harnessed to guide the embodied agents to follow human instructions.
- We introduce a novel approach, MethodName that creates task representations using a pre-trained LLM, without the need for pre-collected datasets or additional training procedures.
- Theoretical analysis and empirical results demonstrate that MethodName effectively generates task-related representations, enabling agents to understand the task, complete the instruction successfully, and outperform the baseline NLC-RL methods.

2 Background

2.1 RL and NLC-RL

We consider environments represented as a finite Markov decision process (MDP) Sutton (2018); Puterman (2014), which is described by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \rho, \mathbf{r}, \gamma)$, where $\mathcal{S}$ represents the state space, $\mathcal{A}$ is the action space. $\mathcal{P}$ represents the probability of transition while ρ represents the initial state distribution. r is a reward function while γ represents the discount factor determining the weights of future rewards. In NLC-RL, the agent receives an NL instruction (L_N) that reflects the human's instruction. The policy $\pi(\cdot|s_t, L_N)$ which is used for decision making is trained conditioned on the state $s_t \in \mathcal{S}$ and language instruction L_N which describes the task (e.g. 'Can you open the light?'). It is crucial to highlight that in our work, we assume no prior knowledge of the exact number of tasks, the relevant task details, or the task instruction descriptions.

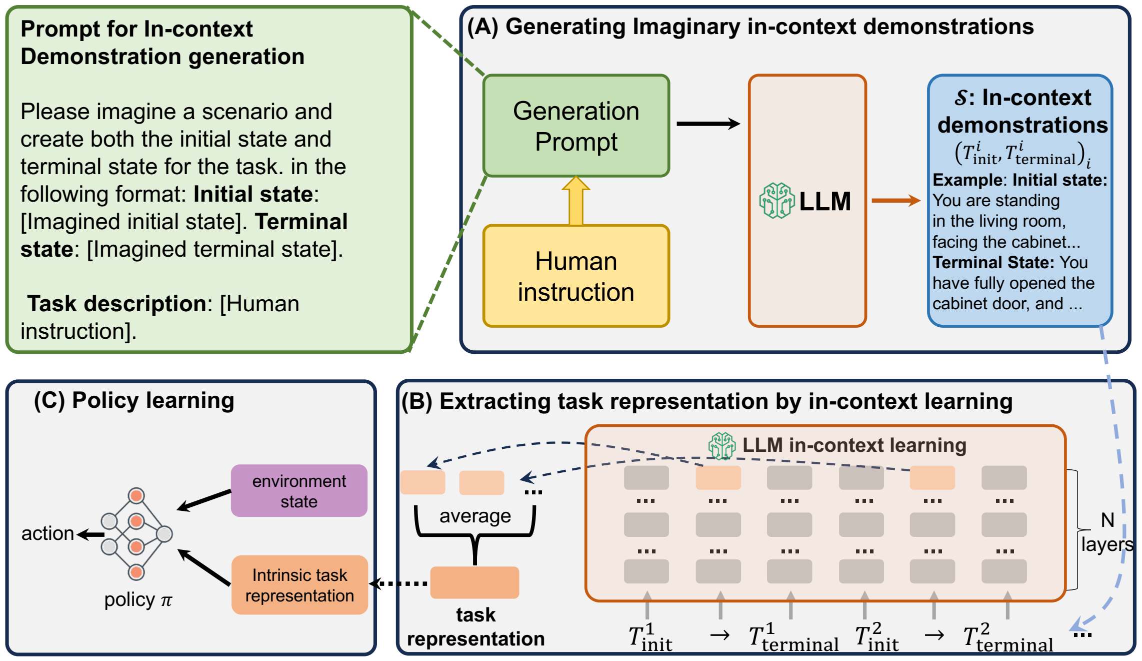

Prompt for In-context Demonstration generation
Please imagine a scenario and create both the initial state and terminal state for the task. in the following format: Initial state: [Imagined initial state]. Terminal state: [Imagined terminal state].
Task description: [Human instruction].
(A) Generating Imaginary in-context demonstrations
Generation Prompt
Human instruction
LLM
S: In-context demonstrations
$(T^i_{\text{init}}, T^i_{\text{terminal}})_i$
Example: Initial state: You are standing in the living room, facing the cabinet... Terminal State: You have fully opened the cabinet door, and ...
(C) Policy learning
environment state
action
policy π
Intrinsic task representation
(B) Extracting task representation by in-context learning
LLM in-context learning
average
N layers
task representation
$T^1_{\text{init}} \rightarrow T^1_{\text{terminal}}$ $T^2_{\text{init}} \rightarrow T^2_{\text{terminal}}$ ...

Figure 1: Overall framework of MethodName method.

The overall objective of NLC-RL is to maximize the expected return under different NL instructions:

$$\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r\left(s_t, a_t, L_N\right) \mid s_0 \sim \rho, a_t \sim \pi\left(\cdot \mid s_t, L_N\right)\right] \tag{1}$$

2.2 ICL and task representations

ICL is a paradigm that enables language models to learn tasks by providing only a few examples as demonstrations Dong et al. (2022); Brown (2020). Formally, given a query input prompt p, a pre-trained language model $\mathcal{M}$ takes the candidate answer conditioned on a demonstration S. S contains k demonstrations, thus $S = \{(x_1, y_1), \cdots (x_k, y_k)\}$. To perform specified tasks for a given query p, the model is asked to predict y based on the demonstrations namely,

$$y = \arg\min \mathcal{M}([S, p]) \tag{2}$$

where $[S, p]$ represents a concatenation of the demonstrations S and input prompt p.

There has been work Hendel et al. (2023) demonstrating that the mechanism behind in-context learning involves using demonstrations to extract a task representation θ. The θ is used to identify the task and generate the output y for the current query p.

2.3 Random Projection Technique for Dimension Reduction

The intrinsic dimensionality of an objective function refers to the minimum number of parameters required to achieve satisfactory solutions (Li et al., 2018). As demonstrated by Aghajanyan et al. (2021), large-scale pre-training empirically compresses the intrinsic dimensionality of downstream tasks. Li et al. (2018) proposes a method to measure this intrinsic dimensionality in neural networks by identifying the minimal subspace dimensionality derived through random projections. This approach to random mapping is rooted in the Johnson-Lindenstrauss lemma (Johnson, 1984), which posits that when points in a vector space are projected onto a randomly chosen subspace of sufficiently high dimension, the distances between points are approximately preserved.

3 Method

Given a task description, we aim to extract a corresponding task representation to guide the agent in completing instruction-following task. The method leverages LLM's internal ability to generate in-context

Algorithm 1 Train Procedure of In-context Learning Embedding as Task Representation (MethodName)

```
Require: LLM $\mathcal{M}$; imaginary in-context demonstrations num $n$; random projection matrix $A \in \mathbb{R}^{k\times d}$; policy update timestep $M$.
Output: instruction following policy $\pi_\phi$;
 1: Initialize parameters of the policy network $\pi_\phi$, and value network $V_\psi$.
 2: for timestep $i = 0$ to $M$ do
 3:     Sample a natural language task instruction $L_N$ from the environment.
 4:     // Generate Imaginary In-context Demonstrations
 5:     for $i = 1$ to $n$ do
 6:         Query LLM $\mathcal{M}$ to generate imaginary in-context demonstrations $(T^i_{\text{init}}, T^i_{\text{terminal}})$ with NL $L_N$.
 7:     end for
 8:     // Extract Task Representation
 9:     Construct mapping $T^i_{\text{init}} \rightarrow T^i_{\text{terminal}}$ with demonstrations.
10:     Extract hidden state with Eq. (3). And fuse it with Eq. (4)
11:     // Train Policy with RL
12:     Use random projection metric to transfer into intrisic task representation $Z_{L_N}$ with Eq. (5).
13:     while episode not terminal do
14:         Observe current state $s_t$.
15:         Execute action $a_t \sim \pi_\phi(\cdot|s_t, Z_{L_N})$, and receive a reward $r_t$ from the environment.
16:     end while
17:     Update the policy $\pi_\phi$ and value functions $V_\psi$ based on the samples collected from the environment.
18: end for
```

demonstrations S. Then we construct task pairs with S and extract a task-specific vector to serve as the conditioning input for policy training.

Our framework, as illustrated in Fig. 1, consists of three stages Specifically. In the first stage, given a task description L_N, we utilize LLM to generate multiple task-relevant <initial state, terminal state> trajectory pair. In the second stage, we concatenate the generated pairs with L_N in the form of *"initial state → terminal state,"* using these as input to the LLM to extract the corresponding hidden states. These hidden states are then fused to form the task representation θ. In the third stage, the high-dimensional task representation is mapped to a lower-dimensional space and used as the condition for policy learning.

3.1 Generating Imaginary In-context Demonstrations by Prompting LLM

To leverage the in-context learning ability of LLMs, the first step is to obtain a set of in-context learning trajectories and construct the demonstration set, which we refer to as *imaginary in-context demonstrations*. MethodName generates imaginary in-context demonstrations by prompting LLMs to imagine specific trajectories (initial/terminal states) corresponding to the human instructions. Given a human instruction x, we utilize the the following prompt p for the input of LLM $\mathcal{M}$: 'Please help me imagine a scenario and create both the initial state and terminal state for the task $\cdots$'. During generation, we create one trajectory at a time, repeating the process n times until the entire trajectory set is generated. We extract the initial state and terminal state from the trajectory to construct the set of imaginary in-context demonstrations S:

$$S = \{(T^1_{\text{init}}, T^1_{\text{terminal}}), \cdots, (T^n_{\text{init}}, T^n_{\text{terminal}})\},$$

where $(T^i_{\text{init}}, T^i_{\text{terminal}}) = \mathcal{M}(x, p)$ is the imaginary in-context demonstrations extracted from the output by the LLM. To ensure that the LLM can generate diverse trajectories for in-context learning, we utilize LLM to sample the output with temperature factor $T = 1$ during the generation process.

The demonstration is structured by presenting the initial state as T_{init} and the terminal state as T_{terminal}. This format is chosen because, when given the task and the initial state T_{init}, the output T_{terminal} from the LLM represents task completion. Namely, $T_{\text{init}} \rightarrow T_{\text{terminal}}$ implicitly signifies the successful completion of the current task, making it easier for us to extract task-related representations.

3.2 Extracting Task Representations via In-context Learning from LLM

In the previous subsection, we introduce how MethodName generates a set of imaginary in-context demonstrations. Now we elaborate on the process for obtaining the task representation that indicates the tasks corresponding to the human instruction. Specifically, we leverage the imaginary in-context demonstrations generated in section 3.1 to perform the task of mapping $T^i_{\text{init}} \rightarrow T^i_{\text{terminal}}$. In particular, the latent state $h^i \in \mathbb{R}^d$ for each $T^i_{\text{init}} \rightarrow T^i_{\text{terminal}}$ is obtained by inputting them into the LLM. Specifically, we extract the hidden state corresponding to the arrow position token and take the hidden state from the final layer of the attention block as the task representation. All the demonstrations are input into LLM total, namely, $\{h_i\}_{i=1}^n \leftarrow \mathcal{M}(\{T^i_{\text{init}} \rightarrow T^i_{\text{terminal}}\}_{i=1}^n)$. However, directly inputting it into the LLM can lead to the following problem: If L_N were placed after the demonstration, the $\rightarrow$ for the first demonstration would only have seen the initial state without being exposed to the task description, leading to an inaccurate hidden state. Therefore, we concatenate L_N before the demonstration S. This method is finally formalized as:

$$\{h_i\}_{i=1}^n \leftarrow \mathcal{M}([L_N, \{T^i_{\text{init}} \rightarrow T^i_{\text{terminal}}\}_{i=1}^n]) \tag{3}$$

For a total of n demonstrations, the extracted hidden states form the set $H := \{h_1, h_2, \cdots, h_n\}$. Finally, we average the hidden states vectors as the final task representation θ:

$$\theta = \frac{1}{n}\sum_{i=1}^{n} h_i \tag{4}$$

Due to the complexity and diversity of language, directly inputs the task description L_N into the LLM $\mathcal{M}$ to obtain the hidden state embedding as task representation struggles to get the exact task representation. In contrast, our in-context learning approach extracts a more specific task representation by leveraging provided demonstrations (Liu et al., 2024), enabling better task representation.

3.3 Policy Learning with Task Representation

In this state, MethodName uses reinforcement learning to train an instruction-following policy (IFP). As the agent collects samples in the environment, the environment randomly generates human language instruction based on the current task. We utilize the method introduced in section 3.1 and section 3.2 to extract task representation. Next, the policy makes decisions for the entire episode based on the current observation and task representation until either the task is completed or the maximum timestep is reached. However, typically the task representation $\theta \in \mathbb{R}^d$ from the LLM is a high-dimensional vector, which is challenging when directly inputting it to the policy. We address this by *random projection* technique, which converts the task representation into a low intrinsic dimensionality, easing the burden on the policy. Specifically, MethodName uses a random matrix $\mathbf{A} \in \mathbb{R}^{k\times d}$ to project the original d-dimensional data, θ, onto a k-dimensional subspace $(k \ll d)$, as expressed in the equation below:

$$Z_k = \mathbf{A}_{k\times d}\theta_d, \tag{5}$$

where Z_k is defined as intrisic task representationand k represents the dimension of subspace. Therefore, we utilize the intrinsic task representation Z as the policy condition input rather than task representation θ. 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 both MethodName and all baselines. During IFP training, the LLM's parameters are frozen. We provide the pseudo-code shown in Algorithm 1 for MethodName to further clarify the process.

4 Theoretical Justification

Let $\mathcal{Z}$ denote the embedding space, and $\mathcal{T}$ denote the task. We assume that the value function can be represented as:

$$V(s) = \phi(s)^T w,$$

where $\phi(s) \in \mathbb{R}^d$ is the representation of the state s in some feature space. This assumption is not too restrictive. When using a neural network to represent the value function, $\phi(s)$ corresponds to the output of the penultimate layer (i.e., the second-to-last layer).

Without loss of generality, we assume:

$$\|\phi(s)\| \leq 1 \quad \text{and} \quad \|w\| \leq W_{\max}.$$

Based on this setup, we aim to minimize the regularized expected risk function:

$$R_{\mathcal{T},\phi}(w) =_{(s_i)\sim\mathcal{T}} \left[\left(\phi(s_i)^T w - y_i\right)^2\right] + \frac{1}{2}\|w\|^2,$$

where $y_i = \sum_{t=0}^{\infty} \gamma^t r_t$ is the cumulative discounted return.

Since computing the full expectation is often infeasible, we instead minimize the regularized empirical risk function:

$$R_D(w) = \frac{1}{n}\sum_{i=1}^{n} \left(\phi(s_i)^T w - y_i\right)^2 + \frac{1}{2}\|w\|^2,$$

where $D = \{(s_i, y_i)\}_{i=1}^{n}$ is a dataset sampled from the distribution $\mathcal{T}$.

Now, when we concatenate the task representation $z \in \mathcal{Z}$ with the state representation, the feature vector becomes $\phi(s, z)$. Thus, the new regularized empirical risk function becomes:

$$R_{D,\mathcal{Z}}(w) = \frac{1}{n}\sum_{i=1}^{n} \left(\phi(s_i, z_i)^T w - y_i\right)^2 + \frac{1}{2}\|w\|^2.$$

Here, $z_i \in \mathcal{Z}$ corresponds to the embedding associated with the state s_i.

In our algorithm, the goal is to solve the following problem:

$$\min_w \mathcal{E}(\mathcal{Z}), \quad \text{where} \quad \mathcal{E}(\mathcal{Z}) =_{(\mathcal{T}_i,z_i)\sim p(\mathcal{T},\mathcal{Z})} {}_{D\sim\mathcal{T}_i} \left[R_{D,\mathcal{Z}}(w)\right].$$

The objective is to find the weights w that minimize the expected risk over all possible tasks $\mathcal{T}$ and task representations $z \in \mathcal{Z}$.

Finally, let $\mathcal{E}^*$ denote the optimal minimum risk. It can be expressed as:

$$\mathcal{E}^* =_{\mathcal{T}\sim p(\mathcal{T})} \left[R_{\mathcal{T},\phi}(w_{\mathcal{T}})\right], \quad \text{where} \quad w_{\mathcal{T}} = \arg\min_w R_{\mathcal{T},\phi}(w).$$

Then we derive the error bound of value function of NLC-RL.

Theorem 1. *Let $(e_i)_{i=1}^{S} \subset \mathbb{R}^S$ be the standard basis and*

$$P \triangleq \mathbb{E}_{i\sim\nu}[e_i e_i^T].$$

We use Φ_z to represent the feature matrix when task representation exists. We assume further that $\Phi_z\Phi_z^\dagger$ is diagonal, then with probability at least $1-\eta$

$$\mathcal{E}(\mathcal{Z}) - \mathcal{E}^* \le \frac{4R_{\max}}{\sqrt{n(1-\gamma)}}\left(W_{\max} + \frac{R_{\max}}{1-\gamma} E_{z\sim p(z)} \operatorname{Tr}(M(z)N(z))\right)^{\frac{1}{2}}$$
$$+ O\left(\sqrt{\frac{1}{2n}\log\left(\frac{2}{\eta}\right) + \frac{d_{\prec(s,z)}}{n}\log\left(\frac{n}{d_{\prec(s,z)}}\right)}\right),$$

where $W_{\max}$, $R_{\max}$, and γ are constants defined in the context, $M(z) =_{\mathcal{T}\sim p(\mathcal{T}|z)} P$, $N(z) =_{\mathcal{T}\sim p(\mathcal{T}|z)} P^\dagger$, and d_ϕ is VC dimension of function space of $\phi(s, z)$.

The proof of this theorem is in Appendix B. The bound presented in Theorem 1 consists of two components. The first component highlights the performance improvement achieved through the additional information provided by task representations. In contrast, the second component accounts for the performance degradation arising from the increasing complexity of the representation function. Therefore, NLC-RL establishes a trade-off between these two components.

For one-hot method, task representation z only corresponds to one task $\mathcal{T}$. Therefore, $p(\mathcal{T}|z)$ is a deterministic distribution. Based on this intuition, the results of one-hot method can be derived as follows. Under the condition of Theorem 1, with probability at least $1-\eta$, the error bound of one-hot method is

$$\mathcal{E}(\mathcal{Z}) - \mathcal{E}^* \le \frac{4R_{\max}}{\sqrt{n(1-\gamma)}}\left(W_{\max} + \frac{R_{\max}}{1-\gamma}\right)^{\frac{1}{2}}$$
$$+ O\left(\sqrt{\frac{1}{2n}\log\left(\frac{2}{\eta}\right) + \frac{d_{\prec(s,z_{\text{one-hot}})}}{n}\log\left(\frac{n}{d_{\prec(s,z_{\text{one-hot}})}}\right)}\right).$$

Similarly, for naive method without NL instruction, the result can be derived by setting $p(\mathcal{T}|z) = p(\mathcal{T})$. Under the condition of Theorem 1, with probability at least $1-\eta$, the error bound of naive method is

$$\mathcal{E} - \mathcal{E}^* \leq \frac{4R_{\max}}{\sqrt{n(1-\gamma)}} \left(W_{\max} + \frac{R_{\max}}{1-\gamma} E_{z\sim p(z)} \operatorname{Tr}(MN) \right)^{\frac{1}{2}}$$
$$+ O\left(\sqrt{\frac{1}{2n} \log\left(\frac{2}{\eta}\right) + \frac{d_{\prec(s)}}{n} \log\left(\frac{n}{d_{\prec(s)}}\right)} \right),$$

where $M =_{\mathcal{T}\sim p(\mathcal{T})} P$ and $N =_{\mathcal{T}\sim p(\mathcal{T})} P^{\dagger}$.

Compared to Theorem 1, one-hot encoding method yields the finest task representation, resulting in the smallest first component but the largest second component. Conversely, the naive method has the largest first component and the smallest second component. NLC-RL, on the other hand, offers a balanced approach. Additionally, Theorem 1 outlines a method for comparing different NLC-RL approaches. Specifically, the less random the distribution $p(\mathcal{T}|z)$ is, the better the resulting error bound. This finding elucidates the characteristics of an effective task representation. In Figure 5, we visualize the results from our method alongside several baseline methods, illustrating why our approach outperforms the others.

5 Experiments

We conduct extensive experiments to evaluate the effectiveness of the proposed MethodName method. Our experiments aim to answer the following important questions:

- How does MethodName perform compared to previous methods on NLC-RL tasks? (Section 5.2)
- How are the task representations generated by MethodName and how do they compare to the task representations from baselines (Section 5.3)?
- Can MethodName be compatible with different large language models? (Section 5.4)
- What is the impact of each component on the overall performance of MethodName? (Section 5.5)

5.1 Experiment Setup

Environments. We conduct experiments on FrankaKitchen (Gupta et al., 2020) and CLEVR-Robot environments (Jiang et al., 2019) as shown in Fig. 2. FrankaKitchen is a multitask environment in which a 9-DoF Franka robot is placed in a kitchen. The goal is to control the robot to interact with the items in order to reach a desired goal configuration. In the environment, we choose four sub-tasks: activate the bottom burner, move the kettle to the top left burner, turn on the light switch, and open the slide cabinet. We treat each sub-task as a different goal configuration. In each trajectory, the environment randomly generates an NL instruction that describes a goal configuration. The language instruction is generated by ChatGPT (Achiam et al., 2023). The CLEVR-Robot environment is designed for agent manipulation in object interaction tasks and is built on the MuJoCo physics engine (Todorov et al., 2012). It contains five different colored balls and an agent (silver point). In each task, the agent moves a ball to achieve a specific position relative to another ball (including four possible positions: front, behind, left, or right) to complete the goal.

Details of natural language instructions. For FrankaKitchen environment, we used ChatGPT to generate 15 different task descriptions for each goal configuration, yielding 60 different NL descriptions. For example, in the task "open the sliding cabinet," one of the descriptions is "Can you please open the sliding cabinet door for me?". In the CLEVR-Robot task, the variation between tasks comes from the different colored balls selected and their relative positions. Therefore, we used 18 natural language sentence patterns to generate different tasks. We fixed the combinations of balls and relative positions, resulting in 8 distinct tasks and a total of 144 distinct NL instructions. An example of NL instruction is "Push the red ball behind the blue ball". We split the total NL instructions into two sets: training and testing set. The training set comprises 10 and 9 different NL instructions for each goal configuration in FrankaKitchen and CLEVR-Robot, respectively. The testing set consists of the remaining NL instructions. During policy training, the agent can only interact with the NL instructions in the training set. The full set of task descriptions can be found in Appendix A.1.

Baselines for comparison. We consider multiple representative methods in NLC-RL that do not train the LLM as baselines: (1). **One-hot** method encodes all natural language instructions (including both

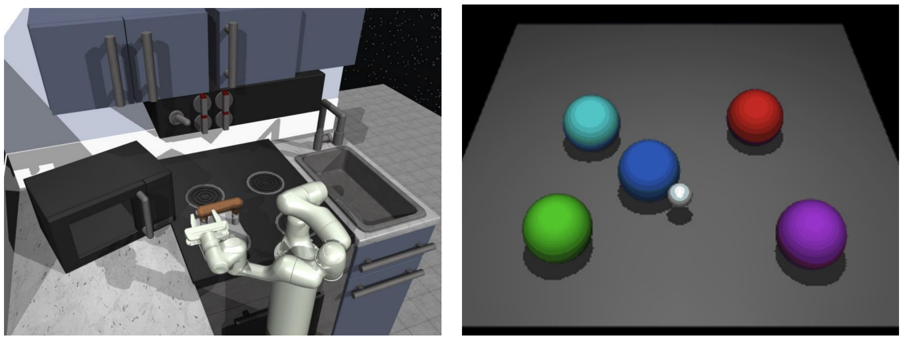
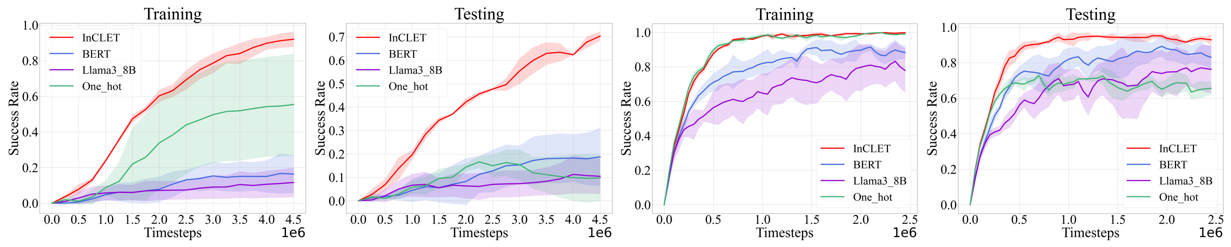

(a) FrankaKitchen (b) CLEVR-Robot

Figure 2: Visualization of the environments in our experiments. (a) FrankaKitchen. The agent controls a 9-DoF Franka robot to manipulate various objects in a kitchen. (b) CLEVR-Robot. The agent (silverpoint) manipulates five different colored balls to reach a specific goal position



(a) Kitchen Training (b) Kitchen Testing (c) Ball Training (d) Ball Testing

Figure 3: Performance of different methods in two environments on training and testing NL instructions. The shaded area stands for the standard deviation across three random trials.

training and testing set) into a one-hot vector. (2). **BERT** method processes all language instructions using a pre-trained BERT model to generate task representations. These representations are then mapped into a continuous space using a fully connected neural network. (3). **Llama3-8B** method is similar to BERT, except that it uses Llama-3.1-8B-Instruct (Dubey et al., 2024) as the pre-trained LLM instead of the BERT model.

Implementation Details. In our experiments, we use Llama-3.1-8B-Instruct (Dubey et al., 2024) to generate imaginary in-context demonstrations and extract task representation. We conducted all experiments with three random seeds, and the shaded area in the figures represents the standard deviation across all three trials. we use the open-sourced RL repository, stable-baselines3 (Raffin et al., 2021) for PPO training. For detailed implementation and hyper-parameters, please refer to Appendix A.1.

5.2 Main Results

Fig. 3 shows the performance of MethodName and baselines on FrankaKitchen and Ball environments. Overall, MethodName outperforms all baselines on both training and testing NL instructions. In the Kitchen environment, baselines that directly employ LLM embedding such as BERT and Llama3-8B struggle to improve the policy performance. This is due to the complexity of diverse NL instructions and the robotic control. The experimental results in Fig. 5 further justify this conclusion. Though one-hot performs well on training NL instructions, it suffers from a significant performance degradation on the test instructions. This can be attributed to that one-hot baseline can only deal with these seen NL instructions, and fail to generalize to unseen instructions. Notably, on these training NL instructions that are unseen during the RL training process, MethodName also obtains a high score. These results highlight MethodName's potential to develop an agent capable of dealing with diverse NL instructions from different scenarios.

5.3 Effectiveness of the MethodName

Previous experiments show MethodName can effectively improve the policy performance compared to the baseline methods. This section investigates the source of such improvement. First of all, we utilize t-SNE projection technique to convert the generated task representation into two-dimensional vectors, as shown in Fig. 5. The points with the same color represent the task representations of the same task, but with different

Task: Activate the bottom burner

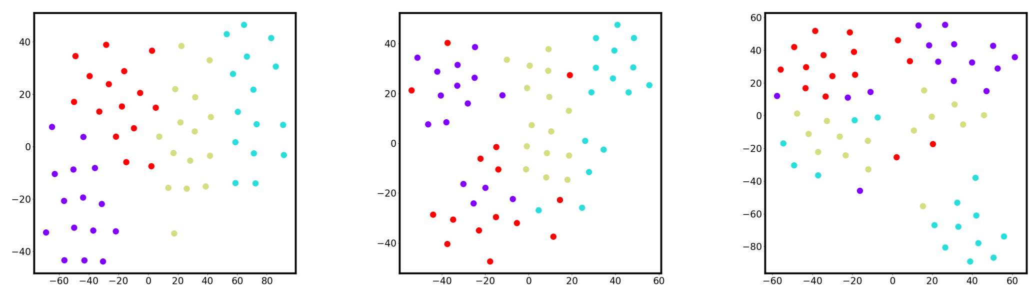
Task Description: Can you turn the knob on the oven that activates the bottom right burner?
Imaginary In-context Demonstrations:
(1). Initial State: The oven is turned off and the kitchen is quiet, with no one around. the knob on the oven that activates the bottom right burner is in the \"off\" position. **Terminal State:** The knob on the oven that activates the bottom right burner is in the \"on\" position, and the burner is heating up with a gentle blue flame. the kitchen is now bustling with activity, with the aroma of baking bread filling the air.
(2). Initial State: The oven door is open, the knob is not turned, and the bottom right burner is off. **Terminal State**: The oven door is open, the knob is turned, and the bottom right burner is on.
(3). Initial State: The oven is turned off, the bottom right burner is cool and dark, and you are holding the oven knob. **Terminal State:** The oven is turned on, the bottom right burner is warm and bright, and the knob is set to the \"on\" position.
(4). Initial State: The person is standing in front of the oven, looking at the knobs on the front, but has not turned any of them yet. the bottom right burner is currently off. **Terminal State**: The person has successfully turned the knob on the oven that activates the bottom right burner, and the burner is now lit.
(5). Initial State: The oven is turned off, the kitchen is quiet, and the user is standing in front of the oven, with their hands free and no objects obstructing their path. **Terminal State**: The oven is turned on, the knob is fully clockwise, the bottom right burner is bright and warm, and the kitchen is filled with the sound of gentle crackling.

Figure 4: Examples of the imaginary in-context demonstrations on FrankaKitchen.

natural language descriptions. We observe that the task representations generated by MethodName get together closer. In contrast, baselines' task representations are more dispersed and scattered. These results justify that MethodName policy can learn more efficiently, as the task representations more accurately capture the meaning of the tasks. Then, we reviewed the examples generated by the LLM, as shown in Fig. 4, which depicts the imaginary demonstrations in the FrankaKitchen environment. We found that the LLM generates demonstrations are highly related to the task description, and these demonstrations cover various scenarios. These demonstrations which are diverse but semantically identical help us extract task representations with the fusion method.

5.4 Compatibility with Different LLMs

We further verify the effectiveness of MethodName method by conducting experiments with different LLMs of various sizes and structures. We conducted experiments with three models in the FrankaKitchen environment and reported their test performance. Specifically, we implement MethodName with Llama3-8B (Dubey et al., 2024), Gemma-7B (Banks and Warkentin, 2024) and Mistral-7B (Jiang et al., 2023), respectively. Additionally, we implemented baseline methods that directly use LLM embeddings for policy training with the models mentioned above. Fig. 6 presents the experimental results. We find that MethodName can effectively improve the NLC-RL performance compared to directly using LLM embeddings as the task representations. Notably, our method delivers consistent performance across different models.



(a) MethodName (b) Baseline (Bert) (c) Baseline (Llama3-8B)
Figure 5: T-SNE projection of the task representations generated by different methods.

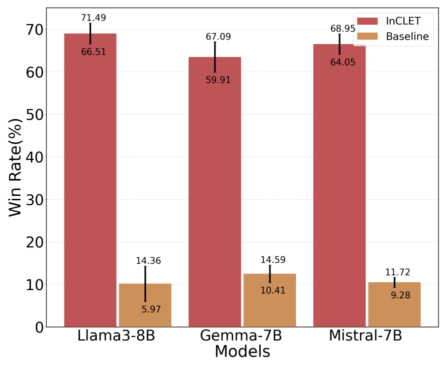


Figure 6: Performance of MethodName with different LLMs on FrankaKitchen environment.

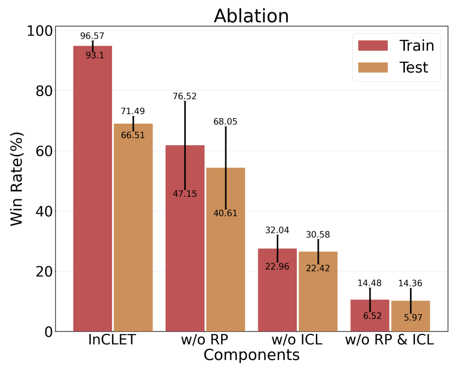


Figure 7: Ablation study on FrankaKitchen environment. The error bar stands for the half standard deviation over three random trials.

5.5 Ablation Study

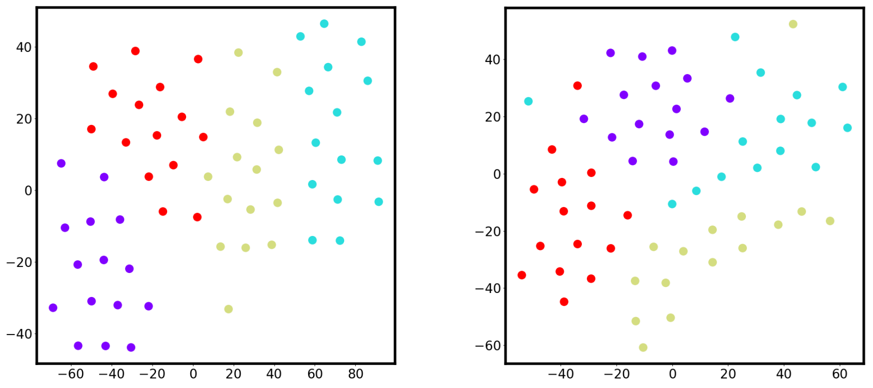
We conduct ablation study to evaluate the impact of each component in MethodName on the overall performance. We consider the following components:in-context learning (**ICL**) that generates task vectors, random projection (**RP**) and different fusion methods.

Ablation study on ICL and RP We conduct ablation studies on the ICL and RP techniques of MethodName method. To implement MethodName w/o ICL, we directly utilize Llama to convert NL instructions into embeddings, and random projection to reduce the dimension. Fig. 7 presents the results of ablation study. The results indicate that both ICL and RP play an important role in MethodName's effectiveness, as MethodName gets the highest scores and MethodName w/o RP & ICL gets the lowest scores. Notably, MethodName w/o RP suffers from a large deviation across different random trials. This could be attributed to that the origin embedding from the LLM could be high-dimensional and complex, highlighting the importance of the dimension reduction by random projection techniques.

Comparing different fusion method. We conducted a fusion-specific ablation to evaluate the impact of the fusion method in MethodName. We conducted a comparative experiment, directly using the hidden state corresponding to $\rightarrow$ in the last demonstration, rather than utilizing the fusion of the hidden states from all demonstrations S. The t-SNE visualization of the representations is shown in the Fig. 8. Experimental results indicate that using only the last hidden state results in less efficient representations. Upon examining the responses shown in Fig. 4, we think this is because the demonstrations imagined by the LLM contain some scene-specific information. By fusing multiple hidden states, we can improve the representation of task-related information.

6 Related Work

Natural Language Conditioned Reinforcement Learning. Natural language conditioned reinforcement learning (NLC-RL) requires agents to complete tasks based on natural language instructions. Previous approaches have mainly focused on two areas. One involves directly using natural language representations as input for the RL policy and training the policy. For example, (Hill et al., 2020) uses a pre-trained language model to encode natural language, and provides the encoded NL as input to the policy. (Misra et al., 2018) learns a policy that maps NL instructions to action sequences. (Chaplot et al., 2018) combines language instruction with observation via Gated-Attention mechanism and then learns policy to execute the instruction. The other involves training an additional encoder, where natural language is first input into the encoder to obtain its representation, which is then used as input for the policy to train the RL model. For example, (Jiang et al., 2019) learns a high-level policy that generates abstract language representations to guide hierarchical RL. (Volovikova et al., 2024) learns a task manager module to handle the language representation for reinforcement learning. Our approach is similar to the first one, which directly uses language embeddings to train the policy. The key difference is that we use in-context learning to extract representations that are better suited for policy learning.



(a) MethodName w/ $\{h_i\}$ fusion (b) MethodName w/ only h_n

Figure 8: T-SNE projection of the task representations generated by different fusion method.

Large Language Model for Reinforcement Learning. Large language models store a wealth of knowledge (Dubey et al., 2024; Touvron et al., 2023; Achiam et al., 2023; Yang et al., 2024), which can be effectively used to guide policy training (Brohan et al., 2023) by aligning the prior knowledge with the knowledge learned by RL. LLMs are leveraged to decompose complex, high-level tasks into multiple low-level, executable step-by-step plans (Volovikova et al., 2024; Huang et al., 2022; Wang et al., 2023b; Brohan et al., 2023). These decomposed plans are then used as conditions for downstream reinforcement learning tasks. However, due to the limited reasoning capabilities of LLMs, they often struggle to deliver precise, effective reasoning. In addition, some works directly use large language models for extracting useful embed information (Myers et al., 2023), generating reward functions (Xie et al., 2024; Pang et al., 2024a), or generating imaginary rollouts for world modeling and policy learning (Pang et al., 2024b). Compared with the above methods, we use large language models as both a generator and task information extractor module. The model automatically generates in-context learning trajectories and uses these to construct demonstrations and extract relevant task information.

In-context Learning. In-context learning (ICL) enables language models to learn tasks given a few demonstrations (Dong et al., 2022) which is similar to the decision process of human beings by learning from analogy (Winston, 1980). The language model is expected to learn the hidden task pattern and accordingly make the right prediction. Therefore, "How to effectively generate high-quality demonstrations" is an important area of research in ICL. Traditional methods heavily rely on human-written demonstrations, which are often limited in quantity, diversity, and creativity. Some work leverages LLM to generate demonstrations (Wang et al., 2023a; He et al., 2024) which is similar to MethodName's trajectories generation (see details in section 3.1). Furthermore, some studies focus on extracting the latent representations from demonstrations (Li et al., 2024) and find that these can provide good task information (Liu et al., 2024; Hendel et al., 2023). Compared to previous methods, our approach does not rely on human-written demonstrations. Instead, MethodNameleverages the internal knowledge of LLMs to generate demonstrations and extract task-specific information. In contrast to (Liu et al., 2024; Li et al., 2024), we validate our method on more complex RL control tasks. It was found that for more complex tasks, effective representations can be extracted.

7 Conclusion

In this paper, we study the integration of LLMs for embodied instruction-following control. We propose a simple and effective approach, MethodName, that processes natural language instruction into task representation. This is the first study investigating the usage of ICL for embodied instruction-following control. Compared to previous NLC-RL methods, MethodName removes the requirements for pre-collecting data from the environment and improves the performance over LLM embeddings. However, there are still some limitations. First, it is challenging to utilize ICL from LLMs to produce effective task representations, when natural language instructions become too complex. When NL instructions are long and detailed, e.g., ‘place the apple on the table, but only after the orange is placed next to the cup’, it is hard to generate the trajectories that capture the conditional or sequential nature of these tasks, leading to failed task representations. Fortunately, the development of LLMs has the potential to mitigate this issue. Second, our method relies on LLMs to generate multiple demonstrations through imagination. If the model is too small, the quality of the generated demonstrations may be sub-optimal. In the future, we will explore the effectiveness of using closed-source models to generate demonstrations in resource-constrained environments, while using smaller models to generate task representations.

References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. (2023). Gpt-4 technical report. *CoRR*, abs/2303.08774.

Aghajanyan, A., Gupta, S., and Zettlemoyer, L. (2021). Intrinsic dimensionality explains the effectiveness of language model fine-tuning. In *ACL/IJCNLP*.

Banks, J. and Warkentin, T. (2024). Gemma: Introducing new state-of-the-art open models. *Google. Available online at: https://blog. google/technology/developers/gemma-open-models/(accessed 6 April, 2024)*.

Brohan, A., Brown, N., Carbajal, J., Chebotar, Y., Dabis, J., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Hsu, J., et al. (2022). Rt-1: Robotics transformer for real-world control at scale. *CoRR*, abs/2212.06817.

Brohan, A., Chebotar, Y., Finn, C., Hausman, K., Herzog, A., Ho, D., Ibarz, J., Irpan, A., Jang, E., Julian, R., et al. (2023). Do as i can, not as i say: Grounding language in robotic affordances. In *CoRL*. PMLR.

Brown, T. B. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Chaplot, D. S., Sathyendra, K. M., Pasumarthi, R. K., Rajagopal, D., and Salakhutdinov, R. (2018). Gated-attention architectures for task-oriented language grounding. In *AAAI*, volume 32.

Dong, Q., Li, L., Dai, D., Zheng, C., Wu, Z., Chang, B., Sun, X., Xu, J., and Sui, Z. (2022). A survey on in-context learning. *CoRR*, abs/2301.00234.

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., et al. (2024). The llama 3 herd of models. *CoRR*, abs/2407.21783.

Gupta, A., Kumar, V., Lynch, C., Levine, S., and Hausman, K. (2020). Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *CoRL*.

He, W., Liu, S., Zhao, J., Ding, Y., Lu, Y., Xi, Z., Gui, T., Zhang, Q., and Huang, X. (2024). Self-demos: Eliciting out-of-demonstration generalizability in large language models. *CoRR*, abs/2404.00884.

Hendel, R., Geva, M., and Globerson, A. (2023). In-context learning creates task vectors. *CoRR*, abs/2310.15916.

Hill, F., Mokra, S., Wong, N., and Harley, T. (2020). Human instruction-following with deep reinforcement learning via transfer-learning from text. *CoRR*, abs/2005.09382.

Huang, W., Abbeel, P., Pathak, D., and Mordatch, I. (2022). Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *ICML*.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., Casas, D. d. l., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., et al. (2023). Mistral 7b. *CoRR*, abs/2310.06825.

Jiang, S., Pang, J., and Yu, Y. (2020). Offline imitation learning with a misspecified simulator. In *NeurIPS*.

Jiang, Y., Gu, S. S., Murphy, K. P., and Finn, C. (2019). Language as an abstraction for hierarchical deep reinforcement learning. *Advances in Neural Information Processing Systems*, 32.

Johnson, W. B. (1984). Extensions of lipshitz mapping into hilbert space. In *CMAP*.

Kenton, J. D. M.-W. C. and Toutanova, L. K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*.

Li, C., Farkhoor, H., Liu, R., and Yosinski, J. (2018). Measuring the intrinsic dimension of objective landscapes. In *ICLR*.

Li, J., Wang, Q., Zhang, L., Jin, G., and Mao, Z. (2024). Feature-adaptive and data-scalable in-context learning. *CoRR*, abs/2405.10738.

Liu, S., Ye, H., Xing, L., and Zou, J. Y. (2024). In-context vectors: Making in context learning more effective and controllable through latent space steering. In *ICML*.

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

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

Myers, V., He, A. W., Fang, K., Walke, H. R., Hansen-Estruch, P., Cheng, C.-A., Jalobeanu, M., Kolobov, A., Dragan, A., and Levine, S. (2023). Goal representations for instruction following: A semi-supervised language interface to control. In *CoRL*. PMLR.

Pang, J., Yang, S., Chen, X., Yang, X., Yu, Y., Ma, M., Guo, Z., Yang, H., and Huang, B. (2023). Object-oriented option framework for robotics manipulation in clutter. In *IROS*.

Pang, J.-C., Wang, P., Li, K., Chen, X.-H., Xu, J., Zhang, Z., and Yu, Y. (2024a). Language model self-improvement by reinforcement learning contemplation. In *ICLR*.

Pang, J.-C., Yang, S.-H., Li, K., Zhang, J., Chen, X.-H., Tang, N., and Yu, Y. (2024b). Knowledgeable agents by offline reinforcement learning from large language model rollouts. In *NeurIPS*.

Pang, J.-C., Yang, X.-Y., Yang, S.-H., Chen, X.-H., and Yu, Y. (2024c). Natural language instruction-following with task-related language development and translation. *NeurIPS*.

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268).

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *CoRR*, abs/1707.06347.

Sun, Y., Toni, L., and Andreopoulos, Y. (2024). Conditional meta-reinforcement learning with state representation. In *ICML AutoRL Workshop*.

Sutton, R. S. (2018). Reinforcement learning: An introduction. *A Bradford Book*.

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

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. (2023). Llama 2: Open foundation and fine-tuned chat models. *CoRR*, abs/2307.09288.

Van der Maaten, L. and Hinton, G. (2008). Visualizing data using t-sne. *Journal of machine learning research*, 9.

Volovikova, Z., Skrynnik, A., Kuderov, P., and Panov, A. I. (2024). Instruction following with goal-conditioned reinforcement learning in virtual environments. *CoRR*, abs/2407.09287.

Wang, Y., Kordi, Y., Mishra, S., Liu, A., Smith, N. A., Khashabi, D., and Hajishirzi, H. (2023a). Self-instruct: Aligning language models with self-generated instructions. In *ACL*.

Wang, Z., Cai, S., Chen, G., Liu, A., Ma, X., Liang, Y., and CraftJarvis, T. (2023b). Describe, explain, plan and select: interactive planning with large language models enables open-world multi-task agents. In *NeurIPS*.

Winston, P. H. (1980). Learning and reasoning by analogy. *Communications of the ACM*, 23(12).

Xie, T., Zhao, S., Wu, C. H., Liu, Y., Luo, Q., Zhong, V., Yang, Y., and Yu, T. (2024). Text2reward: Reward shaping with language models for reinforcement learning. In *ICLR*.

Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., Li, C., Li, C., Liu, D., Huang, F., et al. (2024). Qwen2 technical report. *CoRR*, abs/2407.10671.

A Implementation Details

A.1 Tasks for Evaluation

A.1.1 FrankaKitchen

Task description. The FrankaKitchen environment emulates the realistic kitchen setting with objects that can be interacted with. The 9-DoF Franka robot is required to interact with a burner, a kettle, a light switch, and the slide cabinet in our experiments. The observation space is defined as $\mathbb{R}^{30}$ to represent each object's state. The action space is $\mathbb{R}^{9}$, with 7 DoF for the arm and 2 for the gripper and the actions are represented as the joint velocity. At each timestep, the reward is calculated as the difference between the distance to the target robot's posture at the previous step and the distance to the target robot's posture at the current step. A reward of +1 is given for success, and 0 for failure (Pang et al., 2024c).

Natural Language Instructions. In our experiments, four goal configurations are set up for the robot to execute. We employ ChatGPT to generate natural languages that describe each goal configuration. We ask GPT with prompts like: "I want to ask my robot to open the sliding cabinet door. Please give me 15 different instructions to use." We obtain 15*4=60 different instructions to control four different tasks, here are all the instructions we use in our experiments:

- **–(Bottom Burner)–**
- Can you turn the knob on the oven that activates the bottom right burner?
- Please turn the knob on the oven that activates the bottom right burner.
- Activate the bottom right burner on the oven by turning the knob.
- Would you mind turning the knob on the oven that activates the bottom right burner?
- Turn the knob on the oven to activate the bottom right burner, please.
- Could you turn the knob on the oven so that the bottom right burner activates?
- Robot, please turn the knob on the oven to activate the bottom right burner.
- Activate the bottom right burner by turning the oven knob.
- Robot, turn the knob on the oven that activates the bottom right burner.
- Please turn the knob on the oven to activate the bottom right burner, Robot.
- Would you mind turning the oven knob to activate the bottom burner?
- The bottom burner is activated.
- Rotate the knob located on the oven to start the burner at the bottom right.
- Please turn the oven knob so that the bottom right burner activates, Robot.
- Please activate the bottom right burner on the oven by turning the knob, Robot.
- **–(Switch)–**
- Please turn on the lights
- Can you switch on the lights, please?
- Could you turn on the light switch, please?
- Switch on the lights, please
- Please activate the lighting
- Turn the lights on, please
- Light up the room, please
- Please, turn on the light switch
- Make it brighter in here, can you switch on the lights?
- I need some light, can you switch on the lights?
- Can you switch on the lights in here, please?

- Please, switch on the lights
- Please, turn on the lights in this room
- Turn on the lights, please
- Please switch on the light
- **–(Cabinet)–**
- Can you please open the sliding cabinet door for me?
- Please open the sliding cabinet door for me.
- Can you slide open the cabinet door, please?
- Please open the cabinet door on the sliding mechanism right away.
- Can you open the cabinet door by sliding it?
- Can you open the sliding cabinet door now?
- Please slide open the cabinet door for me now.
- Could you slide the cabinet door to the side?
- Please open the cabinet door by sliding it.
- Please slide open the cabinet door.
- Could you slide the cabinet door open?
- Please, would you mind me opening the slide cabinet door
- I need you to open the sliding cabinet door.
- Can you open the sliding door on the cabinet, please?
- Could you please slide the cabinet door open?
- **–(Kettle)–**
- Please put the kettle on the burner.
- Can you start heating the water in the kettle, please?
- Could you fill the kettle with water and place it on the stove?
- Hey robot, can you make some tea by boiling water in the kettle?
- It's tea time! Could you turn on the stove and place the kettle on top?
- I would like some hot water. Can you put the kettle on the burner, please?
- Robot, please turn on the burner and put the kettle on top.
- I need some hot water. Could you fill the kettle, put it on the stove, and turn on the burner?
- Would you mind putting the kettle on the burner and turning on the stove?
- Can you please prepare some hot water by placing the kettle on the burner and turning on the heat?
- Can you start heating up the water by placing the kettle on the burner and turning on the stove?
- Please start boiling some water in the kettle on the burner, robot.
- Robot, could you put the kettle on the burner and heat up the water?
- Let's make some tea. Can you put the kettle on the burner and turn on the heat?
- Can you please heat up some water by putting the kettle on the burner and turning on the stove?

In each trajectory, there is a randomly selected instruction description for the robot to execute. We test different speaking manners of natural language to prove the effectiveness and robustness of InCLET.

A.1.2 CLEVR-Robot

Task description. In CLEVR-Robot environment, there are five balls with different colors and a robot. The robot needs to manipulate the balls to achieve specific tasks. The observation space is defined as $\mathbb{R}^{10}$ to represent the location of each object. The action space is discrete, with $|\mathcal{A}|$=40 to represent moving a ball in one of the four cardinal directions. At each timestep, the reward is calculated as the difference between the distance to the target position at the previous step and the distance to the target position at the current step. A reward of +100 is given for success and -10 for failure.

Natural Language Instructions. In our experiments, eight different goal configurations are tested. We use different natural language templates to express each goal configuration. For example, if the goal configuration is "put the red ball behind the blue ball". Here are the natural language instructions used in our experiments:

- Push the red ball behind the blue ball
- Can you push the red ball behind the blue ball
- Can you move the red ball behind the blue ball
- Can you keep the red ball behind the blue ball
- Can you help me move the red ball behind the blue ball
- Can you help me keep the red ball behind the blue ball
- Can you help me push the red ball behind the blue ball
- Is the red ball behind the blue ball
- Is there any red ball behind the blue ball
- Move the red ball behind the blue ball
- Keep the red ball behind the blue ball
- The red ball moves behind the blue ball
- The red ball is being pushed behind the blue ball
- The red ball is pushed behind the blue ball
- The red ball is being moved behind the blue ball
- The red ball is moved behind the blue ball
- The red ball was pushed behind the blue ball
- The red ball was moved behind the blue ball

These eight goal configurations are depicted in Table 1. Each goal configuration is to put the first ball in the corresponding position of the second ball.

Color of first ball	Position	Color of second ball
red	behind	blue
blue	to the left of	green
green	in front of	purple
purple	to the right of	cyan
green	behind	blue
purple	to the left of	green
cyan	in front of	purple
blue	to the right of	red

Table 1: Eight tasks in CLEVR-Robot

A.2 Network architecture

We employ the IFP network architecture illustrated in Fig. 9. The natural language instructions would generate demonstrations first, then the LLMs receive generated demonstrations to extract task representations. Then the concatenated vectors(i.e., [state, task representation]) serve as input for the policy and value networks. The difference lies in whether to do the demonstration generation and the way to extract task representations. For all baselines, there is no in-context learning and extraction of task representations. For different ablation experiments, specific extraction methods are applied to the LLMs.

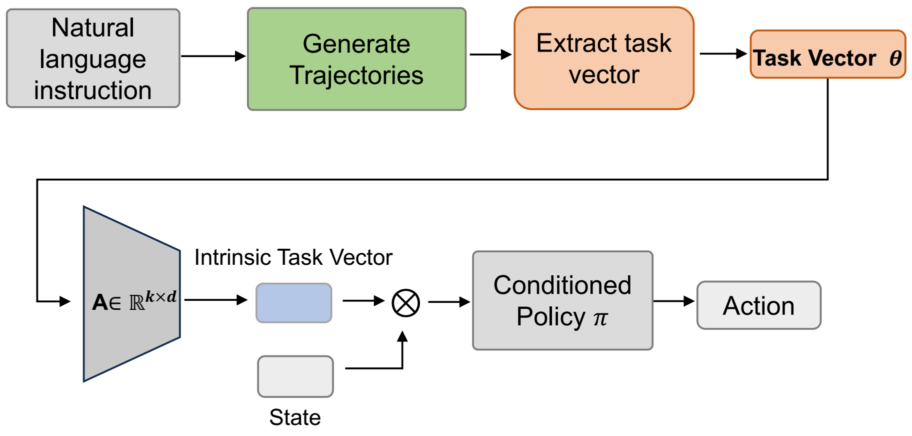


Figure 9: Network architecture of instruction-following policy network in InCLET.

A.3 Prompts for LLM

We should guide the LLM to generate demonstrations with in-context learning value with a given prompt, otherwise, valid task representations cannot be extracted from hidden states. The following is an example of how the prompt guides the model to generate effective demonstrations.

Generating Imaginary in-context demonstrations

Natural Language Instruction: Can you turn the knob on the oven that activates the bottom right burner?
Prompted Task: I'll provide you with a task description. Please help me imagine a scenario and create both the initial state and terminal state for the task. Please help me generate 1 pair of initial state and terminal state for the task using the following format:
Initial state: [your imagined initial state]
Terminal state: [your imagined terminal state].
The task is: Can you turn the knob on the oven that activates the bottom right burner?
Please provide only one example in your response.
LLM: Initial state: The oven is turned off and the kitchen is quiet, with no one around. the knob on the oven that activates the bottom right burner is in the \"off\" position.
Terminal state: The knob on the oven that activates the bottom right burner is in the \"on\" position, and the burner is heating up with a gentle blue flame. the kitchen is now bustling with activity, with the aroma of baking bread filling the air.
Formatted effective demonstrations: The kitchen is dimly lit, and the oven is off. There is a knob on the front of the oven, but its position cannot be determined. There is no food or object obstructing the knob.->The oven is on, and the burner on the bottom right is lit. The knob on the oven is in the activated position. The kitchen is filled with the familiar smell of the oven being turned on.
This procedure will repeat n rounds.

Use prompt to extract task representation

```
Add the in-context prompt to the beginning of the demonstrations:
There is a task:  Can you turn the knob on the oven that activates the
bottom right burner?.  Based on the given four examples, generate the
corresponding terminal states for this task.

The oven is turned off and the kitchen is quiet, with no one around.  the
knob on the oven that activates the bottom right burner is in the \"off\"
position.-> The knob on the oven that activates the bottom right burner
is in the \"on\" position, and the burner is heating up with a gentle
blue flame.  the kitchen is now bustling with activity, with the aroma of
baking bread filling the air.

The oven door is open, the knob is not turned, and the bottom right
burner is off.-> The oven door is open, the knob is turned, and the
bottom right burner is on.

The oven is turned off, the bottom right burner is cool and dark, and
you are holding the oven knob.-> The oven is turned on, the bottom right
burner is warm and bright, and the knob is set to the \"on\" position.

The person is standing in front of the oven, looking at the knobs on the
front, but has not turned any of them yet.  the bottom right burner is
currently off.-> The person has successfully turned the knob on the oven
that activates the bottom right burner, and the burner is now lit.

The oven is turned off, the kitchen is quiet, and the user is standing
in front of the oven, with their hands free and no objects obstructing
their path.->
```

A.4 Hyper-Parameters

The hyper-parameters for implementing MethodNameare presented in Table 2. When implementing baseline methods, we use the same hyper-parameters of PPO for policy learning.

Hyper-parameters	FrankaKitchen	CLEVR-Robot
Shot num	5	5
Intrinsic task representation dimensionality	16	16
Policy LR	3e-4	3e-4
Value LR	3e-4	3e-4
Policy network	[32, 64, 64], tanh	[32, 64, 64], tanh
Value network	[32, 64, 64], tanh	[32, 64, 64], tanh
Mini-batch size	128	128
Nums of mini-batch	160	160
Entropy coefficient	0.1	0.1
Gradient Step	4.5M	2.5M

Table 2: Hyper-parameters.

B Missing Proof

Let $A(\phi(s,z),D)$ be the derived w based on algorithm A, representation $\phi(s,z)$ and dataset D. For any $(\mathcal{T},z) \sim p(\mathcal{T},\mathcal{Z})$, we do the following decomposition:

$$\mathcal{E}_p(\mathcal{Z}) - \mathcal{E}^* = \mathbb{E}_{(\mathcal{T},z)\sim p(\mathcal{T},\mathcal{Z})}[B_{\mathcal{T},z} + C_{\mathcal{T},z} + E_{\mathcal{T},z}].$$

Here,

$$\begin{aligned} B_{\mathcal{T},z} &= \mathbb{E}_{D\sim\mathcal{T}}\left[R_{\mathcal{T},\hat{\phi}}(A(\hat{\phi}(s,z),D)) - R_{\mathcal{T},\phi^*}(A(\phi^*(s,z),D))\right] \\ C_{\mathcal{T},z} &= \mathbb{E}_{D\sim\mathcal{T}}\left[R_{\mathcal{T},\phi^*}(A(\phi^*(s,z),D)) - R_{D,\mathcal{Z}}(A(\phi^*(s,z),D))\right] \\ E_{\mathcal{T},z} &= \mathbb{E}_{D\sim\mathcal{T}}\left[R_{D,\mathcal{Z}}(A(\phi^*(s,z),D)) - R_{\mathcal{T},\phi^*}(w_{\mathcal{T}})\right]. \end{aligned}$$

According to Proposition 1 of (Sun et al., 2024), ϕ^* satisfies

$$(\Phi_z^*)(\Phi_z^*)^T = \frac{\sqrt{n}R_{\max}N(z)^{\dagger/2}\left(N(z)^{1/2}M(z)N(z)^{1/2}\right)^{1/2}N(z)^{\dagger/2}}{4\left(R_{\max}+W_{\max}(1-\gamma)\right)},$$

where Φ_z^* is the feature matrix.

And

$$\begin{aligned} &{}_{(\mathcal{T},z)\sim p(\mathcal{T},\mathcal{Z})}[C_{\mathcal{T},z}, + E_{\mathcal{T},z}] \leq \\ &\frac{4R_{\max}}{\sqrt{n(1-\gamma)}}\left(W_{\max} + \frac{R_{\max}}{1-\gamma}E_{z\sim p(z)}\operatorname{Tr}(M(z)N(z))\right)^{\frac{1}{2}} \end{aligned}$$

Therefore, we only need to bound $B_{\mathcal{T},z}$. Note that $R_{\mathcal{T},\hat{\phi}}(A(\hat{\phi}(s,z),D)) \leq R_{\mathcal{T},\hat{\phi}}(A(\phi^*(s,z),D))$, we have

$$B_{\mathcal{T},z} \leq \mathbb{E}_{D\sim\mathcal{T}}\left[R_{\mathcal{T},\hat{\phi}}(A(\phi^*(s,z),D)) - R_{\mathcal{T},\phi^*}(A(\phi^*(s,z),D))\right] \tag{6}$$

The above equation can be seen as optimizing ϕ with respect to a fixed w, i.e., $A(\hat{\phi}(s,z),D)$. According to the results of learning theory, with probability at least $1-\eta$,

$$\begin{aligned} &\mathbb{E}_{D\sim\mathcal{T}}\left[R_{\mathcal{T},\hat{\phi}}(A(\phi^*(s,z),D)) - R_{\mathcal{T},\phi^*}(A(\phi^*(s,z),D))\right] \leq \\ &O\left(\sqrt{\frac{1}{2n}\log\left(\frac{2}{\eta}\right) + \frac{d_{\prec}}{n}\log\left(\frac{n}{d_{\prec}}\right)}\right). \end{aligned}$$

C Additional Experiment Results

C.1 Examples of Synthetic Demonstrations

In demonstration generation phase, we use LLMs to generate synthetic demonstrations from natural language instructions formatted with prompts, Fig. 10 to Fig. 17 are some examples of synthetic demonstrations.

C.2 Results of T-SNE Projection on CLEVR-Robot

This section investigates the source of improvement on CLEVR-Robot. We utilize t-SNE projection technique to convert the generated task representation into two dimensional vectors, as shown in Fig. 18. The points with same color represent the task representations of a same task, but with different natural language descriptions. We observe that the task representations generated by MethodName get together closer. In contrast, baselines' task representations are more dispersive and scattered. These results justify that MethodName policy can learn more efficiently, as the task representations more accurately capture the meaning of the tasks.

C.3 Train with Oracle

Fig. 20 and Fig. 19 show the performance of MethodName and baselines on FrankaKitchen and CLEVR-Robot environments. We trained an oracle model to determine the performance limits of the model. The oracle model can access the ground truth one-hot vector as the task representation. Overall, except for oracle, MethodName outperforms all baselines on both training and testing NL instructions. In Kitchen environment, baselines that directly employ LLM embedding such as BERT and Llama3-8B struggle to improve the policy performance. Specifically, in the CLEVR-Robot environment, our method performs nearly on par with the oracle, indicating that our approach successfully captures task-relevant representations

Task: Activate the bottom burner

Task Description: Can you turn the knob on the oven that activates the bottom right burner?
Imaginary In-context Demonstrations:
(1). Initial State: The oven is turned off and the kitchen is quiet, with no one around. the knob on the oven that activates the bottom right burner is in the \"off\" position. **Terminal State:** The knob on the oven that activates the bottom right burner is in the \"on\" position, and the burner is heating up with a gentle blue flame. the kitchen is now bustling with activity, with the aroma of baking bread filling the air.
(2). Initial State: The oven door is open, the knob is not turned, and the bottom right burner is off. **Terminal State**: The oven door is open, the knob is turned, and the bottom right burner is on.
(3). Initial State: The oven is turned off, the bottom right burner is cool and dark, and you are holding the oven knob. **Terminal State:** The oven is turned on, the bottom right burner is warm and bright, and the knob is set to the \"on\" position.
(4). Initial State: The person is standing in front of the oven, looking at the knobs on the front, but has not turned any of them yet. the bottom right burner is currently off. **Terminal State**: The person has successfully turned the knob on the oven that activates the bottom right burner, and the burner is now lit.
(5). Initial State: The oven is turned off, the kitchen is quiet, and the user is standing in front of the oven, with their hands free and no objects obstructing their path. **Terminal State**: The oven is turned on, the knob is fully clockwise, the bottom right burner is bright and warm, and the kitchen is filled with the sound of gentle crackling.

Figure 10: Examples of Task 'Activate the bottom burner' on FrankaKitchen.

Task: Switch on the lights

Task Description: Can you switch on the lights, please?
Imaginary In-context Demonstrations:
(1). Initial State: The room is dark, and you are standing in the corner of the room, unable to see much of anything. **Terminal State:** The room is now well-lit, with soft, warm lighting that illuminates the entire space, making it easy for you to see and move around.
(2). Initial State: The room is dimly lit, with only a faint glow from the window. The light switches are located on the walls, but they are all in the \"off\" position. **Terminal State**: The room is fully lit, with bright overhead lighting and soft, warm ambient lighting from table lamps. The light switches are all in the \"on\" position, and the room is now bright and welcoming.
(3). Initial State: The room is dark, with only faint moonlight shining through the windows, and the lights are off, making it difficult for anyone in the room to see. **Terminal State:** The room is fully lit, with bright and warm light streaming from the newly switched-on light fixtures, allowing everyone in the room to see clearly and comfortably.
(4). Initial State: The living room is dark, and the light switch is turned off. **Terminal State**: The living room is brightly lit, and the light switch is turned on.
(5). Initial State: The room is dark, and there is no light. The light switch is located near the door, but it is not touched yet. **Terminal State**: The room is now lit, with a warm and cozy glow. The light switch is in the \"on\" position, and the light bulbs are emitting a soft, gentle light that illuminates the entire room.

Figure 11: Examples of Task 'Switch on the lights' on FrankaKitchen.

Task: Open the sliding cabinet door

Description: Please open the sliding cabinet door for me.
Imaginary In-context Demonstrations:
(1). Initial State: The sliding cabinet door is closed, and you are standing in front of the cabinet. **Terminal State*:*** The sliding cabinet door is open, and you are able to access the contents of the cabinet.
(2). Initial State: The sliding cabinet door is closed, and you are standing in front of it, with the door handle in your hand, the door is in the \"closed\" position and you can't see what's inside the cabinet. **Terminal State**: The sliding cabinet door is now fully open, and you can see the contents of the cabinet, it is in the \"fully open\" position and you are holding the handle, you have a clear view of the items inside the cabinet.
(3). Initial State: The sliding cabinet door is closed, the kitchen is clean, and the user is standing in front of the cabinet. **Terminal State:** The sliding cabinet door is fully open, the user is standing inside the cabinet, and the kitchen utensils and other items are visible and accessible.
(4). Initial State: The sliding cabinet door is securely shut, and it's hard to tell if it's locked or not, but i'm standing in front of it, looking at it, and thinking about how to open it. **Terminal State**: The sliding cabinet door is now fully open, revealing the contents of the cabinet, and i'm easily able to reach in and grab what i need.
(5). Initial State: The sliding cabinet door is shut, and the user is standing in front of it, with their hand extended towards the door. **Terminal State**: The sliding cabinet door is open, and the user is easily able to access the contents of the cabinet.

Figure 12: Examples of Task 'Open the sliding cabinet door' on FrankaKitchen.

Task: Put the kettle on the burner

Task Description: Robot, could you put the kettle on the burner and heat up the water?
Imaginary In-context Demonstrations:
(1). Initial State: The robot is sitting idle in the kitchen, the kettle is placed on the countertop, the burner is turned off, and the water in the kettle is cold. **Terminal State*:*** The robot is successfully heating up the water in the kettle, the kettle is now placed on the burner, the burner is turned on, and the water in the kettle is steaming hot.
(2). Initial State: The robot is standing on the kitchen counter next to the sink, the kettle is not plugged in, and the burner is off. **Terminal State**: The robot has successfully placed the kettle on the burner and turned it on, and the water inside the kettle has started to boil.
(3). Initial State: The robot is in the kitchen, the kettle is currently off the burner, the water is at room temperature, and the stove is turned off. **Terminal State:** The robot has successfully put the kettle on the burner, the stove is turned on, and the water has begun to heat up.
(4). Initial State: The robot is sitting on the kitchen counter, the kettle is placed on the counter in front of it, the burner is turned off, and the water in the kettle is at room temperature. **Terminal State**: The kettle is placed on the burner, the burner is turned on, and the water in the kettle is hot and ready to be used for making tea or coffee.
(5). Initial State: The kettle is placed on the table, the stove is off, and the water in the kettle is at room temperature. **Terminal State**: The kettle is on the stove, the burner is lit, and the water inside the kettle has been heated up to boiling point.

Figure 13: Examples of Task 'Put the kettle on the burner' on FrankaKitchen.

Task: Push the red ball behind the blue ball

Task Description: Push the red ball behind the blue ball.
Imaginary In-context Demonstrations:
(1). <u>Initial State:</u> A blue ball is on the table and a red ball is in front of it.
<u>Terminal State:</u> The red ball is pushed behind the blue ball.
(2). <u>Initial State:</u> The red ball is on the left side of the playing area, and the blue ball is on the right side of the playing area. There are no other balls or obstacles in the playing area.
<u>Terminal State</u>: The red ball is behind the blue ball in the playing area, indicating that the red ball has been successfully pushed behind the blue ball. There are still no other balls or obstacles in the playing area.
(3). <u>Initial State:</u> The red ball is placed on the table in front of the blue ball. the blue ball is placed on the edge of the table, and the red ball is near the edge of the table. both balls are stationary. **<u>Terminal State:</u>** The red ball is pushed behind the blue ball and is stationary at that position. the blue ball remains stationary.
(4). <u>Initial State</u>: The red ball is in front of the blue ball, with a gap between them, and both balls are moving towards the right edge of the table. **<u>Terminal State</u>**: The red ball is behind the blue ball, with no gap between them, and both balls have come to a stop at the right edge of the table.
(5). <u>Initial State</u>: A red ball is in the middle of a track, and a blue ball is also in the middle of the track, but slightly in front of the red ball. **<u>Terminal State</u>**: The red ball is behind the blue ball.

Figure 14: Examples of Task 'Push the red ball behind the blue ball' on CLEVR-Robot.

Task: Push the blue ball to the left of the green ball

Task Description: Can you push the blue ball to the left of the green ball.
Imaginary In-context Demonstrations:
(1). <u>Initial State:</u> The blue ball is 5 spaces to the right of the green ball, and both balls are resting on a long flat table.
<u>Terminal State:</u> The blue ball is right next to the green ball, also resting on the table.
(2). <u>Initial State:</u> The blue ball and the green ball are on a table, with the blue ball slightly to the right of the green ball.
<u>Terminal State</u>: The blue ball is now on the left side of the green ball, having been pushed in that direction.
(3). <u>Initial State:</u> The blue ball is to the right of the green ball on the table. The table is clear of any other objects.
<u>Terminal State:</u> The blue ball is to the left of the green ball on the table. The table is still clear of any other objects.
(4). <u>Initial State</u>: The blue ball is on the right side of the screen, and the green ball is in the middle of the screen. the blue ball is static, and the green ball is also static.
<u>Terminal State</u>: The blue ball is to the left of the green ball, and both balls are static. The blue ball has reached the leftmost edge of the screen, and the green ball is at the middle of the screen.
(5). <u>Initial State</u>: The blue ball is located right next to the green ball, and both balls are initially in the starting line.
<u>Terminal State</u>: The blue ball is on the left side of the green ball in the desired final location.

Figure 15: Examples of Task 'Push the blue ball to the left of the green ball' on CLEVR-Robot.

Task: Put the green ball in front of the purple ball

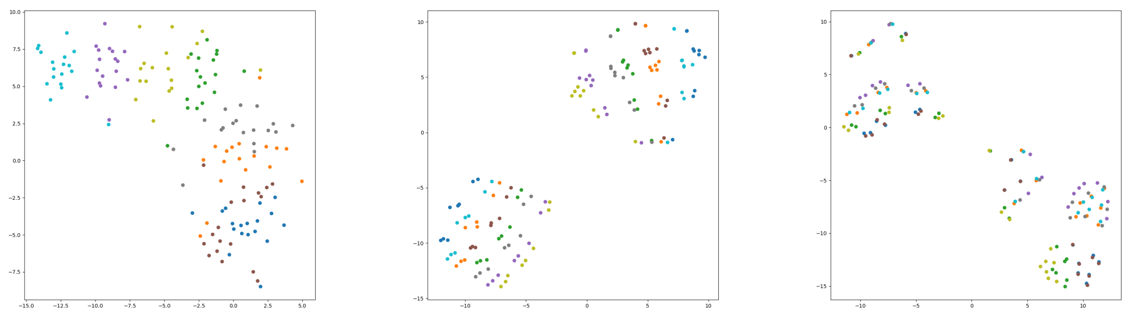
Task Description: Move the green ball in front of the purple ball.
Imaginary In-context Demonstrations:
(1). Initial State: The green ball is placed 5 units to the left of the purple ball, and both balls are stationary on a flat surface. **Terminal State*:*** The green ball is now situated directly in front of the purple ball, both still on the flat surface but now with the green ball positioned in a straight line with the purple ball.
(2). Initial State: The green ball and purple ball are on the same table, but the green ball is behind the purple ball. The table is empty, except for the two balls. **Terminal State**: The green ball is now in front of the purple ball, with a small gap between them on the table.
(3). Initial State: There are two balls on a table, a green ball is on the far left side of the table and a purple ball is on the far right side of the table, the distance between them is approximately 10 inches.
Terminal State: The green ball is in front of the purple ball.
(4). Initial State: The green ball is in the bottom-left corner of the table and the purple ball is in the top-right corner.
Terminal State: The green ball is in the front of the purple ball. This means the green ball is positioned in the middle of the table between the purple ball and the bottom edge. the purple ball should remain in the top-right corner.
(5). Initial State: The green ball is on the left side of the table, and the purple ball is on the right side of the table, separated by some distance.
Terminal State: The green ball is directly in front of the purple ball, side by side on the same side of the table.

Figure 16: Examples of Task 'Put the green ball in front of the purple ball' on CLEVR-Robot.

Task: Put the purple ball to the right of the cyan ball

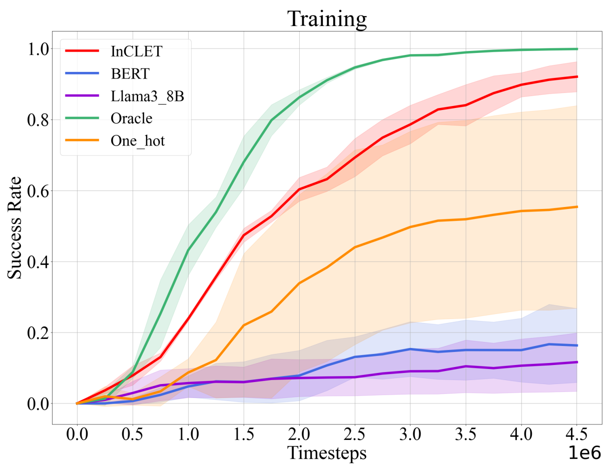
Task Description: Keep the purple ball to the right of the cyan ball.
Imaginary In-context Demonstrations:
(1). Initial State: The purple ball is on the left side of the table and the cyan ball is on the right side of the table.
Terminal State*:* The purple ball is on the right side of the table and the cyan ball is on the left side of the table.
(2). Initial State: The purple ball is on the left of the cyan ball. **Terminal State**: The purple ball is on the right of the cyan ball.
(3). Initial State: There is a table with two balls on it. The purple ball is on the left side and the cyan ball is on the right side. **Terminal State:** The purple ball is on the right side of the table, and the cyan ball is on the left side.
(4). Initial State: The purple ball is to the left of the cyan ball and on the table.
Terminal State: The purple ball is to the right of the cyan ball and on the table.
(5). Initial State: The purple ball is on the left side of the board and the cyan ball is on the right side of the board.
Terminal State: The purple ball is on the right side of the board and the cyan ball is on the left side of the board.

Figure 17: Examples of Task 'Put the purple ball to the right of the cyan ball' on CLEVR-Robot.

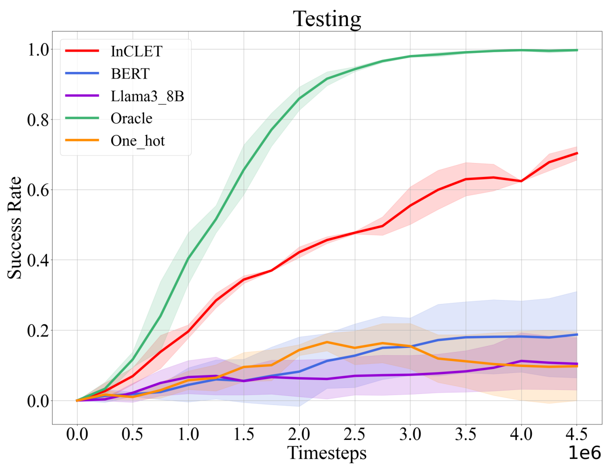


(a) MethodName (b) Baseline (Bert) (c) Baseline (Llama3-8B)

Figure 18: T-SNE projection of the task representations generated by different methods.

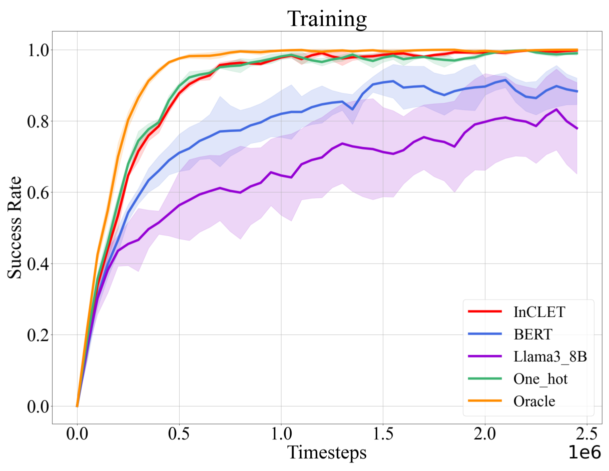


(a) Kitchen Training

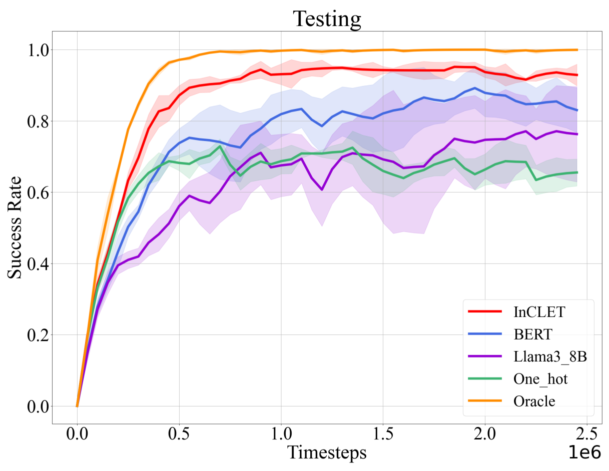


(b) Kitchen Testing

Figure 19: Performance of different methods in FrankaKitchen on training and testing NL instructions. The shaded area stands for the standard deviation across three random trials.



(a) Ball Training



(b) Ball Testing

Figure 20: Performance of different methods in CLEVR-Robot on training and testing NL instructions. The shaded area stands for the standard deviation across three random trials.