VQ-CNMP: Neuro-Symbolic Skill Learning for Bi-Level Planning

Hakan Aktas
Department of Computer Science and Technology
The University of Cambridge
United Kingdom
hea39@cam.ac.uk

Emre Ugur
Department of Computer Engineering
Bogazici University
Turkey
emre.ugur@bogazici.edu.tr

Abstract: This paper proposes a novel neural network model capable of discovering high-level skill representations from unlabeled demonstration data. We also propose a bi-level planning pipeline that utilizes our model using a gradient-based planning approach. While extracting high-level representations, our model also preserves the low-level information, which can be used for low-level action planning. In the experiments, we tested the skill discovery performance of our model under different conditions, tested whether Multi-Modal LLMs can be utilized to label the learned high-level skill representations, and finally tested the high-level and low-level planning performance of our pipeline.

Keywords: Skill Discovery, Neuro-Symbolic Robotics, Multi-Level Planning, Task and Motion Planning

1 Introduction

Planning in the continuous real-world domain is a hard problem. Using state [1, 2, 3], and action abstractions [4] has been shown to address this difficulty by separating high-level decision-making from low-level perception and control, where abstractions showed great success in making fast and effective robotic plans that can be generalized to other environments. Our paper, on the other hand, focuses on using learned action and state predicates as an intermediary layer between virtual generalist agents and environments. It is difficult to learn from low-level continuous robotic data as it is complex and noisy, especially in the real world. Furthermore, the data is limited as it is not mostly produced by people and is only collected in controlled lab settings. These two issues make it hard to include raw robotic data in training datasets of multi-purpose large models such as LLMs. However, since their training data (text) includes information about numerous domains, LLMs have (although limited) reasoning capabilities in multiple domains, including reasoning for high-level robotic tasks. We believe abstractions can be used to bridge this gap between real-world environments and generalist virtual agents. State abstraction methods can be the eyes, and action abstractions can be the arms and legs of these embodied virtual generalist agents. In the case of LLMs, Instead of including the robotic data in the training data of the LLM, one can train small abstraction models that could bridge the gap between the agent and its embodiment and the environment in which the embodiment resides. This

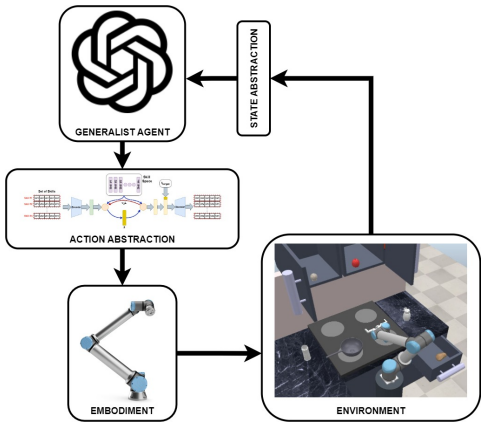

Figure 1: An overview of a generalist agent that uses abstractions to interact with its environment.

way, these abstraction models can handle all the low-level information while the agent is capable of high-level reasoning, which can be learned from text much more efficiently. An overview of how a system like this would function is provided in Figure 1.

[5, 4] showed that it is sufficient to learn symbolic categories for the precondition and the effects for long-horizon planning. [6] built upon this framework, focusing on sharing agent-centric symbols between different tasks. Combining traditional clustering and classification machine learning algorithms, [7, 8] discovered object-centric precondition and effect symbols for generating PDDL descriptions. These works commonly cluster the collected random interactions based on actions and effects prior to the symbol learning procedure. Our work studies learning of discrete skills from demonstrations for long-horizon planning with LLM. In another line work, [9, 10, 11] proposed a bi-level planning schema in which a set of operators and corresponding samplers were previously acquired, allowing the agent to make refined plans. [3] learned symbolic predicates from demonstrations optimizing planning performance. These studies follow the task and motion planning (TAMP) formulation [12] in which the problem is finding the sequence of tasks and the correct motion parameters. These TAMP formulations learn state abstractions on top of high-level predicates such as on(?x, ?y). Utilizing recent and advanced neural network architectures, [2] learned symbols with an encoder-decoder network to minimize effect prediction error for probabilistic PDDL-based planning for single and paired objects. Extending their previous work, [13, 14, 15] utilized the attention layer to learn relational symbols in environments with varying numbers of objects and used these symbols in PDDL-based planning. In our work, we aim to benefit from LLMs for long-horizon planning, benefiting from the impressive capabilities of the visual-language foundational models.

In this paper, we propose a novel action abstraction model that can learn discrete high-level skill representations from demonstrations that include low-level variations of skills. It is important to emphasize that high-level skills are learned without losing the low-level information. Importantly, our model ıs capable of clustering skills in unevenly mixed datasets in an unsupervised manner. We also propose a bi-level planning pipeline that an LLM agent can use to execute high-level plans. After discovering the high-level skills, the model can also be utilized to make low-level plans using a gradient-based approach. The proposed pipeline includes the following steps:

1. Demonstrations are clustered, and high-level skills are discovered using our model.
2. Discovered skills are labeled either by an expert or a Multi-Modal LLM.
3. A new model is trained using self-supervision provided by the discovered high level skill labels.
4. An external agent is used to make high-level plans using the discovered skills for given goal tasks
5. Low-level execution plans are made for each step of the high-level plan using our proposed gradient-based planning approach.

Our contributions are as follows:

1. A novel skill discovery method that can learn high-level skill representations from an unlabeled demonstration data
2. A bi-level planning pipeline based on our proposed model

2 Method

2.1 Model

In this study, we propose a novel neural network model that can learn discrete latent representations from skill demonstrations. The model utilizes the vector quantization approach [16] to learn single discrete vectors from different variations of a high-level skill. For example, taking a sandwich from the refrigerator is a high-level skill, while where in the refrigerator the sandwich is taken from

constitutes the variation or the low-level component of the high-level skill. Given a demonstration dataset that includes different variations of multiple high-level skills, using a vector-quantized autoencoder architecture, different skills get assigned to different vectors in the vector space as training progresses. While learning the skill vector space, the model also learns a distribution for each vector which spans the low-level planning space of each high-level skill. Furthermore, our model is also capable of grouping unlabeled demonstrations into skills since the given demonstrations are given without labels and order.

We used the architecture of the Conditional Neural Movement Primitives [17] to leverage its ability to learn latent spaces from continuous movement trajectories and incorporated vector quantization to learn a discrete vector space from given demonstrations. More formally, during training, given a set of demonstrations $D = (t, SM(t))$ where $SM(t)$ denotes sensorimotor information at time t, at each training iteration a demonstration trajectory, D_j is randomly sampled from the dataset. From this trajectory, a set of points are sampled which can be denoted as $(t_i, SM(t_i))_{i=0}^{i=n}$ and are fed through the encoder,

$$z_i = E((t_i, SM(t_i))|\theta) \quad (t_i, SM(t_i)) \in D_j \tag{1}$$

where z_i denotes the latent representation generated by the encoder E with parameters θ using data point $(t_i, SM(t_i))$. Then these representations are averaged,

$$z_e = \frac{1}{n}\sum_{i=0}^{n} z_i \tag{2}$$

to get a single latent representation z_e. Then similar to [16], the representation is passed through a discretization bottleneck by mapping it to the nearest vector in the skill space,

$$z_q = v_k \quad where \quad k = argmin_m(||z_e - v_m||) \tag{3}$$

where v_m denotes the m^{th} vector in the space. Then the resulting representation z_q is concatanated with the target time t_{target} (the time step of the outputted data point) and passed through the decoder to generate the output,

$$(\mu_{t_{target}}, \sigma_{t_{target}}) = Q((z_q, t_{target})|\phi) \tag{4}$$

where ϕ denotes the parameters of the decoder Q, $\mu_{t_{target}}$ denotes the mean and the $\sigma_{t_{target}}$ denotes the variance of the output. The model overview can be seen in Figure 2. The loss of the model is defined as the following,

$$Loss = -\log P(SM(t_{target})|\mu_{t_{target}}, \sigma_{t_{target}}) + ||sg(z_e) - z_q||_2^2 + \beta * ||z_e - sg(z_q)||_2^2 \tag{5}$$

which includes both the reconstruction loss (negative-log likelihood) of the CNMP (the term on the left) and the loss terms of Vector Quantization (on the middle and the right). sg denotes the stop gradient operator. Unlike [16], we observed that using β values that are not small causes the gradients to explode. We used $\beta = 0.25$ in our experiments and observed that β values as small as 1 were enough to cause the gradients to explode.

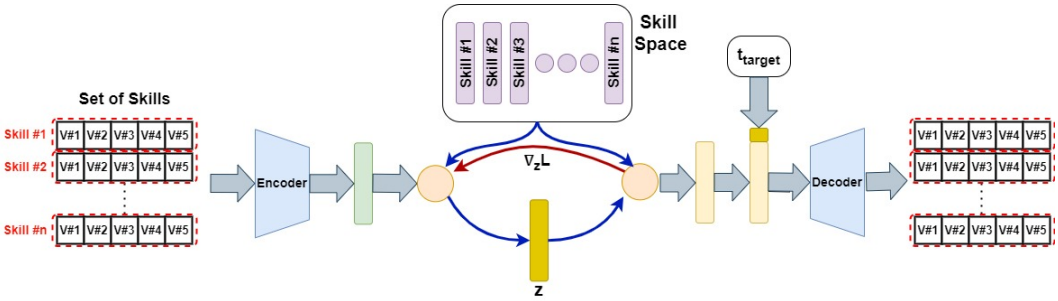

Figure 2: The Overview of our Model

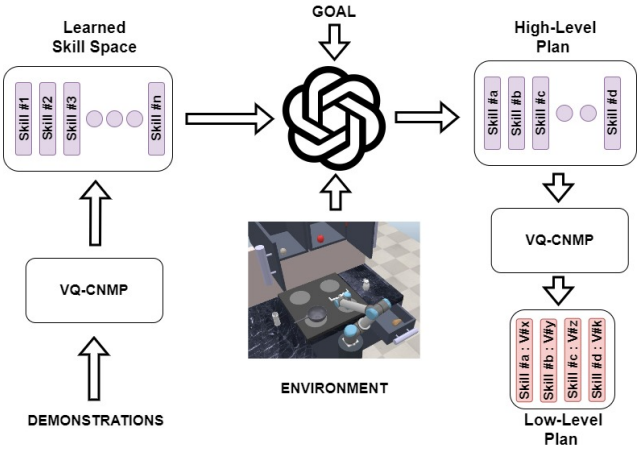

Figure 3: Planning system

After the training is complete, the learned skill space includes one vector for each skill in the demonstration dataset used during training, and the action trajectories generated using these vectors are general representations of these skills. Since each skill is a distribution of low-level variations, the high-level general representations can be thought of as the means of these distributions. Labeling these vectors (what each skill is) can be done by hand (by watching the execution), but we also tried using ChatGPT-4o for this process to automate the pipeline of our system as much as possible.

2.2 Self-Supervised Fine-Tuning

While the model can successfully cluster the demonstrations into discrete skills, our experiments showed that the actions the system can generate are not reliable enough to be used in low-level plans. We propose self-supervised fine-tuning using the skills the system discovers to solve this issue. After the initial training is complete and the skills are discovered, i.e., the demonstrations are clustered into skills, training a new model using the discovered skills significantly increases the low-level planning performance. To achieve this, instead of letting the system decide which vector to choose by Euler distance in the vector quantization layer, we used the discovered clusters to assign the demonstrations that were assigned to the same vector in the first training phase to the same vector..

2.3 Planning

The overview of the planning system can be seen in Figure 3. For the high-level plan, after the skill space is learned and labeled, the actions are given in a prompt along with the goal and the image of the environment. Since we are not using state abstractions in this work, we used a Multi-Modal LLM (ChatGPT-4o) that can make inferences for given images. The prompt also includes the statement that only the given actions can be used, and an ordered list of these actions should be returned.

For the low-level plan, we used a gradient-based method similar to the one utilized in [18]. The method uses high-level skill vectors as inputs and passes them through the decoder of the trained model to get an initial prediction of the low-level action. Then, at each iteration, gradient descent is used to get the initial prediction closer to the goal by adjusting the input vector. For practicality, we used end-effector pose data and gripper state as input to the model for the planning experiments, so that the part of the action prediction where the robot makes contact with the object would be close to the object's location which is what differs between the low-level versions of the same high-level skill. For each step of the high-level plan, the vector of that step's skill is given as input to the decoder of our model, along with the target time close to the time the robot makes contact with the object it is interacting with. Since the change between the variations of the skills is where they retrieve the objects from, the time step that the robot makes contact with the object will be close to

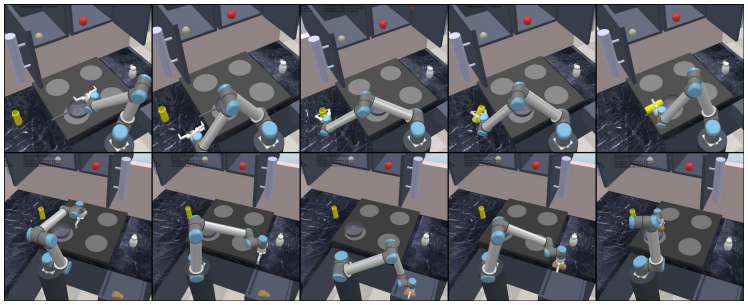

Figure 4: Adding oil and potato to the pan actions.

the initial location of the object for that generated trajectory. After that, the difference between the predicted object location and the actual object location is calculated using MSE;

$$loss = MSE((x_{\mu_{e_t}}, y_{\mu_{e_t}}, z_{\mu_{e_t}}), (x_{object}, y_{object}, z_{object})) \tag{6}$$

The result is used as loss, and the gradient of the input vector is taken and applied. Note that the model's parameters are unchanged in this step, and only the vector given as input to the decoder is adjusted. This process is repeated until the predicted object location is sufficiently close (empirically set, based on the object, for all less than 2 cm). After the process is complete, the adjusted vector is fed into the system's decoder along with all time steps to generate the action, which is the low-level variation of this skill to achieve this step of the high-level plan. This process is repeated for each vector in the high-level plan.

3 Experimental Results

To test the capabilities of our model, we created a kitchen environment. We selected cooking as the general task and conceptualized skills that would complement it, such as taking ingredients from the cupboards or drawers and dropping them into the pan, taking the oil bottle near the stove and adding oil to the pan, taking the salt near the stove and adding it to the pan where we tried to utilize objects that are frequently used in the real world. Some of these skills can be seen in Figure 4. We collected 100 demonstrations for each skill by changing the objects' locations within reachable/valid positions.

3.1 Skill Discovery

To test the skill discovery performance of our model, we trained 100 randomly initialized models and analyzed them in batches of 10. The results showed that in 27 of all trials, the skills were clustered perfectly into the 5 vectors that were in the high-level skill space. We further analyzed them in batches of 10 to find a practical way of selecting a model that automatically clustered the demonstrations. In all batches, we observed that two of the three models with the least valued losses in the batch were perfectly clustered. Generally, when the model could not perfectly cluster the skills, this was because multiple skills were mapped to the same vector.

We also investigated how the model behaves when the size of the skill space is more or less than the number of skills in the demonstrations. We set the size of the skill space to 3, 10, 20, and 100,

	Skill Space Size				
	3	5	10	20	100
perfect clustering	73/100	27/100	30/100	32/100	16/100
Accuracy	0.98292	0.97	0.95876	0.94720	0.89894

Table 1: Demonstration Clustering results. Perfect clustering represents the number of models where demonstrations of each skill were assigned to the same vector. Accuracy measures the percentage of demonstrations of the same skill assigned to the same vector.

	1	2	3	4	5	6	7	8	9	10	random
picking up tomato	%33	%29	%62	%64	%27	%60	%67	%63	%71	%50	%70
potato	%20	%7	%15	%25	%45	%40	%33	%63	%100	%100	%83
salt	%7	%29	%23	%33	%54	%30	%33	%38	%57	%67	%44
oil	%20	%43	%23	%50	%54	%30	%75	%38	%29	%67	%65
mushroom	%20	%14	%23	%25	%9	%40	%33	%38	%85	%50	%48

Table 2: LLM labeling accuracy results. The first ten columns show the prediction accuracy when one to ten consecutive snapshots are used, and the last column shows the accuracy when a random number of randomly selected snapshots are used.

trained 100 models each, and evaluated them the same way. Results showed that when the size of the vector space is 3, the model clustered some of the skills together, and out of the 100 models 73 of them were perfectly clustered (the demonstrations that belong to the same skill were clustered to the same vector). We also observed that when the skill space is smaller, the combined loss values of the models were significantly higher (on average, 4 vs 6 between the models with the least combined loss). When the skill spaces were larger than the skills in the demonstration set, we observed that 30, 32, and 16 of the 100 models were clustered perfectly for the models with the vector spaces of size 10, 20, and 100, respectively. When we examined the demonstration distribution of each model, we observed that the model divided the demonstrations that belong to the same skill into different vectors, reducing the vector quantization loss. The minimum observed vector quantization loss was around 0.7, 0.02, 0.01, and 0.002 for vector space sizes 5, 10, 20, and 100, respectively. In practice, to find a good estimation of the number of skills in the demonstration dataset, one can train several batches of models with varying vector space sizes and compare the vector quantization losses. We also calculated an accuracy value by setting the correct label of each skill to the vector to which the highest number of demonstrations are mapped. As seen in Table 1, as the number of vectors in the vector space increases, the accuracy values drop, meaning the demonstrations belonging to the same skill get distributed to multiple vectors, which, as stated, reduces the vector quantization losses.

Another concern when using demonstration datasets that include multiple skills is bias. While we used an equal number of demonstrations per skill to train the model for each result, we also experimented with unequal mixes. We randomly sampled a random number of demonstrations (no less than 15 for each) per skill, which showed that the demonstration clustering accuracy was not significantly affected by inequalities in the data mixes. In each 100-model batch, around 30 were perfectly clustered.

Based on the results, in practice, our model can be utilized by training a batch of models with significantly larger skill spaces than the number of skills in the demonstration dataset, and the ones with the least losses can be used. The vectors that represent the same action can be detected and labeled by executing the actions generated by the decoders using the vectors.

3.2 Labeling using Multi-Modal LLMs

To test whether Multi-Modal LLMs can be used for labeling skills, we took image snapshots of the execution of the action generated using the vectors in the learned skill space at different time steps. We gave a prompt to ChatGPT-4o asking what the robot might be trying to achieve along with different combinations of the snapshots. All demonstrations used during the experiments have a size of 150, and we took snapshots every 10 data points. Results can be seen in Table 2. The first 10 columns show the prediction accuracy when one to ten consecutive images are used for prediction. We also experimented with a random number of randomly selected snapshots from the actions seen in the last column. While a general increase in accuracy can be observed as the number of snapshots increases, in some cases, it also caused the LLM to be distracted from the skill, which resulted in predictions that are more about the environment. When we examined the results of both consecutive and random cases, we observed that the predictions were more successful when snapshot combinations that sufficiently demonstrated the skill were used, which ideally equals a number of images before making contact with the interacted object, an image of the contact and several images after making contact. The use of more images only deteriorated the prediction performance.

3.3 Planning Performance

3.3.1 High-Level Planning Performance

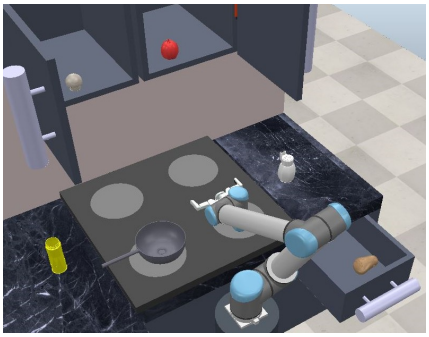

Figure 5: Image of the environment used for high-level planning.

For high-level planning, we selected the tasks of making stews that contain different ingredients. Since results in Subsection 3.2 showed that using LLMs for labeling is not very reliable, we labeled each skill by hand. We described the skill space in the prompt as retrieving the objects from their positions and adding them to the pan without stating where each object is. and gave the goals similar to "Given the robotic environment in the image, how can the robot make a stew made of potato, mushroom, and salt?". We also gave an image of the environment as seen in Figure 5 . During testing, we included 10 trials from each possible combination of the ingredients (310 in total). The high-level planning performance of ChatGPT-4o can be seen on the left column of the Table 3.

3.3.2 Effect of Number of Skills in the Skill Space

Since the "distractions" in the environment significantly affected the LLM performance in Subsection3.2, we also tested whether irrelevant information about skills affects the high-level planning performance. To test the effect, we checked three conditions: environment-related actions, environment-unrelated actions, and both. Environment-related actions are relevant to the current setting, such as opening/-closing cupboards/drawer and moving the pan to the other stoves. Unrelated actions are irrelevant to the current environment, such as plugging in the phone charger or moving the chair closer to the table. The results can be seen in the Table 3. As seen from the second, third, and last columns, the planning performance drops when high-level skill space includes unneeded skills. The results also showed that performance was affected slightly more by the irrelevant skills.

Table 3: High-level planning results of ChatGPT-4o.

	only skills	with relevant actions	with irrelevant actions	both
planning performance	%59.67	%49.67	%46.77	%48.71

3.3.3 Environment with Hidden Elements

One limitation of using the environment image only is that the image may not contain all the information needed to achieve the tasks. Humans also share this limitation, which is overcome by having prior information about the environment or exploring it. We closed the cupboards and drawer to test whether we could achieve the former. We used the image in Figure 6 and included the whereabouts of the objects and the skills for opening and closing the cupboards/drawers to the prompts. Previously, we did not include opening and closing actions to the data mix we used to train our model since their variations are insignificant in execution. With the closed state of the environment and locations of the hidden objects given in the prompt, the system achieved a high-level planning performance of %77.74. When we also included the locations of the other two objects (oil and salt), the planning per-

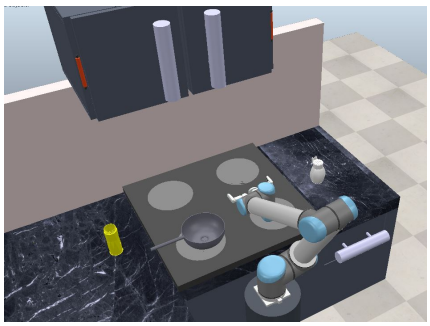

Figure 6: An image of the environment with hidden objects used for high-level planning.

		tomato	mushroom	potato	salt	oil
self-supervised	pick-up	%100	%100	%90	%100	%100
	put in pan	%90	%85	%80	%85	%80
unsupervised	pick-up	%80	%40	%0	%40	%90
	put in pan	%0	%0	%0	%0	%0

Table 4: Comparison of our model's low-level single-task planning performance when trained self-supervised vs unsupervised.

formance rose to %98.38, which shows that the visual understanding of the LLM is unreliable for this setting. Although we did not use them in this study, this result also shows the importance of state abstractions.

We also tested the exploration case. When we did not explicitly state that it could explore the environment first, it made assumptions about the places of the objects in the cupboards and the drawer. It made the plan directly, which resulted in an incorrect outcome. However, when explicitly stated, it first planned actions to open the cupboards and drawer, asked for an updated image of the environment, and made the plan according to the updated environment. Based on all the experiments, it can be stated that the performance of the LLM depends heavily on the quality of the given prompt and the understanding of the environment, which would have to be optimized for the environment. While a general prompt that performs well in every environment might be engineered, the reliability of generalizing to every possible environment would still be questionable.

3.3.4 Low-Level Planning Performance

Since the skills used in the experiment are mostly independent of each other (the success of one does not depend on the success of the others), we evaluated our model's low-level performance on single-task trials. Since our planning approach functions by bringing the step of the action trajectory close to the object position, we examined each task in two parts: picking up the objects and adding them to the pan. We also compared the low-level planning performance of the model when it is trained self-supervised and unsupervised. The results can be on the Table 4. It can be seen from the Table that the model that trained using self-supervision significantly outperformed the unsupervised one. While the actions produced by the unsupervised model were not precise enough to complete the whole task, in most cases, they could pick up the object since that is what the gradient-based planner optimizes the trajectories for. Another difference we observed between the two models is that, on average, the self-supervised model took significantly fewer iterations to optimize the high-level skill (70 for self-supervised and 400 for unsupervised).

4 Conclusion, Limitations, Future Work and Discussion

In this paper, we propose a novel neural network model capable of clustering skills in mixed datasets and can be used for bi-level planning. We tested its ability to discover skills under different conditions, such as uneven dataset mixes and varying sizes of the skill space. We also tested whether LLMs can be used to label the discovered skills. Lastly, we show our system's planning performance. One limitation of our model is that demonstrations must follow the same distribution to be classified as having the same skill. Furthermore, since the system depends on an external agent for reasoning and we don't use state abstractions, its planning performance highly depends on the external agent's perception and high-level planning capabilities. Finally, while the environment image is enough to have information about the initial state for high-level planning, the exact locations of the objects to be interacted with are needed to make the low-level plans. In future work, we plan on experimenting with other LLMs to test their high-level planning and skill labeling performance. We also plan on comparing our approach to other models using larger datasets.

References

[1] A. Ahmetoglu, B. Celik, E. Oztop, and E. Ugur. Discovering predictive relational object symbols with symbolic attentive layers. *IEEE Robotics and Automation Letters*, 2024.

[2] A. Ahmetoglu, M. Y. Seker, J. Piater, E. Oztop, and E. Ugur. Deepsym: Deep symbol generation and rule learning for planning from unsupervised robot interaction. *Journal of Artificial Intelligence Research*, 75:709–745, 2022.

[3] T. Silver, R. Chitnis, N. Kumar, W. McClinton, T. Lozano-Pérez, L. Kaelbling, and J. B. Tenenbaum. Predicate invention for bilevel planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12120–12129, 2023.

[4] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez. From skills to symbols: Learning symbolic representations for abstract high-level planning. *Journal of Artificial Intelligence Research*, 61: 215–289, 2018.

[5] G. Konidaris, L. Kaelbling, and T. Lozano-Perez. Constructing symbolic representations for high-level planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 28, 2014.

[6] S. James, B. Rosman, and G. Konidaris. Learning portable representations for high-level planning. In *International Conference on Machine Learning*, pages 4682–4691. PMLR, 2020.

[7] E. Ugur and J. Piater. Bottom-up learning of object categories, action effects and logical rules: From continuous manipulative exploration to symbolic planning. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2627–2633. IEEE, 2015.

[8] E. Ugur and J. Piater. Refining discovered symbols with multi-step interaction experience. In *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*, pages 1007–1012. IEEE, 2015.

[9] T. Silver, R. Chitnis, J. Tenenbaum, L. P. Kaelbling, and T. Lozano-Pérez. Learning symbolic operators for task and motion planning. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3182–3189. IEEE, 2021.

[10] R. Chitnis, T. Silver, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling. Learning neuro-symbolic relational transition models for bilevel planning. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4166–4173. IEEE, 2022.

[11] T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling. Learning neuro-symbolic skills for bilevel planning. In *Conference on Robot Learning (CoRL)*, 2022.

[12] C. R. Garrett, R. Chitnis, R. Holladay, B. Kim, T. Silver, L. P. Kaelbling, and T. Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4(1):265–293, 2021.

[13] A. Ahmetoglu, E. Oztop, and E. Ugur. Learning multi-object symbols for manipulation with attentive deep effect predictors. *arXiv preprint arXiv:2208.01021*, 2022.

[14] A. Ahmetoglu, E. Oztop, and E. Ugur. Symbolic manipulation planning with discovered object and relational predicates. *arXiv preprint arXiv:2401.01123*, 2024.

[15] A. Ahmetoglu, B. Celik, E. Oztop, and E. Ugur. Discovering predictive relational object symbols with symbolic attentive layers. *arXiv preprint arXiv:2309.00889*, 2023.

[16] A. Van Den Oord, O. Vinyals, et al. Neural discrete representation learning. *Advances in neural information processing systems*, 30, 2017.

[17] M. Y. Seker, M. Imre, J. H. Piater, and E. Ugur. Conditional neural movement primitives. In *Robotics: Science and Systems*, volume 10, 2019.

[18] H. Aktas, U. Bozdogan, and E. Ugur. Multi-step planning with learned effects of partial action executions. *Advanced Robotics*, 38(8):562–576, 2024.

A Experimental Results

A.1 Labeling Using Multi-Modal LLMs

The prompt we used to generate labels for the snapshots can be seen below.

```
Prompt: I will provide you with snapshot images of a robot
   interacting with a single object. Your task is to analyze
   each image and predict the robot's specific action based
   solely on visual cues. Return a concise description of the
   action that best fits the images. Provide only the action
   description with no additional text or explanations.
```

A.2 Planning

A.2.1 High-Level Planning Performance

The prompt template we used to make the high-level plans can be seen below. The $< ingridients >$ part is changed between trials, and the ingredients are given as a comma-separated list.

```
Prompt: Given the robotic environment in the image, how can the
    robot make a stew using the following ingredients: <
   ingredients>? The available actions are:
{ 1:"retrieve the object in the right cupboard and add to the
   pan", 2:"retrieve the object in the left cupboard and add
   to the pan", 3:"retrieve the object in the drawer and add
   to the pan", 4:"retrieve the object on the left of the
   stove and add to the pan", 5:"retrieve the object on the
   right of the stove and add to the pan ", }
Select the necessary actions to achieve the goal and return
   only their corresponding keys in a list, formatted as JSON.
    No explanation, descriptions, or additional output is
   needed.
```

A.2.2 Effect of Number of Skills in the Skill Space

The skill list used for the relevant actions part can be seen below.

```
    1: retrieve the object in the right cupboard and add it to
       the pan,
    2: retrieve the object in the left cupboard and add it to
       the pan,
    3: retrieve the object in the drawer and add it to the pan,
    4: retrieve the object on the left of the stove and add to
       the pan",
    5: retrieve the object on the right of the stove and add it
        to the pan,
    6: put the pan to top right stove,
    7: put the pan to the top left stove,
    8: put the pan on the bottom right stove,
    9: put the pan to the bottom left stove,
    10: close the right cupboard,
    11: close the left cupboard,
    12: close the drawer,
    13: open the drawer,
```

```
    14: open the right cupboard,
    15: open the left cupboard,
    16: open the oven,
    17: start the blender,
    18: close the microwave,
    19: open the microwave,
    20: start the microwave at high heat,
    21: start the microwave at low heat,
    22: stop the microwave,
    23: plug in the microwave,
```

The skill list used for the irrelevant actions part can be seen below.

```
    1: retrieve the object in the right cupboard and add it to
       the pan
    2: retrieve the object in the left cupboard and add to the
       pan
    3: retrieve the object in the drawer and add to the pan
    4: retrieve the object on the left of the stove and add to
       the pan
    5: retrieve the object on the right of the stove and add to
        the pan
    6: charge the phone
    7: plug in the microwave
    8: plug in the phone charger
    9: open the door
    10: close the door
    11: synchronize the clock
    12: fill the glass with water
    14: wipe the floor
    15: mop the floor
    16: vacuum the living room
    17: vacuum the dining room
    18: vacuum the bedroom
    19: vacuum the cellar
    20: turn on the lights
    21: turn off the lights
    22: charge the laptop
    23: put the kids to sleep
    24: wash the clothes
    25: start the washing machine
    26: start the dryer
    27: dry the clothes
    28: hang the clothes
    29: make the beds
    30: clean the desk
```

A.2.3 Environment with Hidden Elements

The prompt used for the part with prior knowledge can be seen below.

```
Prompt: Given the robotic environment in the image, how can the
    robot make a stew using the following ingredients: <
   ingredients>? The tomato is located in the right cupboard,
   the mushroom in the left cupboard, and the potato in the
   drawer. The available actions are: <actions>. Return only
   the keys of the actions necessary to complete the task in
   the correct sequence, formatted as a list in JSON. No
   additional text or explanations are required.
```

The prompt used for the part with exploration can be seen below.

```
Prompt: Given the robotic environment in the image, how can the
    robot make a stew made of <ingredients>? Only the
   following actions can be used: <actions>. You are free to
   explore the environment using the given actions and ask for
    an updated version of the environment.
```