Neuro-Symbolic Imitation Learning: Discovering Symbolic Abstractions for Skill Learning

Leon Keller[1,H,*], Daniel Tanneberg[H], and Jan Peters[1,2,3]

***Abstract*— Imitation learning is a popular method for teaching robots new behaviors. However, most existing methods focus on teaching short, isolated skills rather than long, multi-step tasks. To bridge this gap, imitation learning algorithms must not only learn individual skills but also an abstract understanding of how to sequence these skills to perform extended tasks effectively. This paper addresses this challenge by proposing a neuro-symbolic imitation learning framework. Using task demonstrations, the system first learns a symbolic representation that abstracts the low-level state-action space. The learned representation decomposes a task into easier subtasks and allows the system to leverage symbolic planning to generate abstract plans. Subsequently, the system utilizes this task decomposition to learn a set of neural skills capable of refining abstract plans into actionable robot commands. Experimental results in three simulated robotic environments demonstrate that, compared to baselines, our neuro-symbolic approach increases data efficiency and improves generalization capabilities. Moreover, the learned symbols are interpretable.**

I. Introduction

Over the last decade, imitation learning [1]–[5] has emerged as a powerful approach for teaching new behaviors to robots. Although existing approaches excel at teaching isolated skills, real-world tasks often involve multiple steps that require combining a variety of skills. Thus, to effectively employ robots in our daily lives, we require algorithms capable of utilizing demonstrations to teach not only individual skills but also how these skills can be sequenced to solve extended, intricate tasks [6]–[8]. For instance, consider a robot designed to assist in the kitchen: To be a truly useful assistant, the robot must not only be able to execute individual skills – such as placing a pot on the stove, adding water, and turning on the heat – but also understand how to sequence these skills correctly to cook a complete meal.

Humans have a remarkable ability to tackle such long, complex tasks by relying on a fundamental cognitive tool: abstraction [9]–[11]. We naturally simplify and distill information, recognize patterns, and abstract away noise and unnecessary information. This process not only aids in comprehending what we see and experience but also serves as the foundation for reasoning about our surroundings, planning complex behavior, and generalizing knowledge from one context to another. Similarly, when programming robots to solve complex tasks, human engineers often rely on hierarchical approaches, one of the most popular being Task and Motion Planning (TAMP) [12]–[14]. TAMP involves using a high-level symbolic abstraction of the state-action space to generate task plans, which are subsequently refined into actionable low-level commands through classical motion planning. While this approach is effective, it requires considerable effort from a human expert: Engineers typically must manually design symbolic representations for high-level planning and develop models for low-level motion planning.

In this work, we propose a neuro-symbolic imitation learning framework that learns individual skills and planning capabilities from task demonstrations. The neuro-symbolic policies, akin to TAMP, consist of symbolic components for abstract planning and neural components for refining abstract plans into actionable commands. While previous work explored learning symbols when skills are given [15]–[35], and skills when symbols are given [36]–[47], our approach aims at acquiring symbols *and* skills from raw task demonstrations.

We demonstrate the advantages of our neuro-symbolic framework by comparing it to baselines in three simulated robotic environments. Experimental results show that our neuro-symbolic approach increases data efficiency, improves generalization capabilities, and facilitates interpretability.

II. Neuro-Symbolic Policies

In our framework, policies have both symbolic and neural components. The symbolic components consist of predicates $\mathcal{P}$ that abstract the state-space and operators Σ that define a transition model in the abstract state-space induced by the predicates. Together, predicates and operators define a planning problem in the Planning Domain Definition Language (PDDL) [48] and can be utilized to generate abstract plans. The neural components consist of skills Π that together enable the execution of abstract plans in the environment.

1) Predicates: A predicate $p \in \mathcal{P}$ is a binary function with typed parameters Θ that specifies a relation between a number of objects. For instance, the predicate $\mathrm{onTop}(\mathrm{cube}, \mathrm{cube})$ specifies the onTop relation between objects of type cube. A grounded predicate $\hat{p}$ is the truth value of a predicate when evaluated for specific objects, e.g., for given objects $o_1,\ o_2$ with $t(o_1) = t(o_2) = \mathrm{cube}$ the grounded predicate $\mathrm{onTop}(\mathrm{o_1}, \mathrm{o_2}) = \mathrm{True}$ specifies that o_1 is on top of o_2. Given an environments state $\boldsymbol{s} \in \mathcal{S}$ and a set of predicates $\mathcal{P}$, we define the corresponding abstract state $\bar{\boldsymbol{s}} = \psi(\boldsymbol{s}, \mathcal{P})$ as the set of all grounded predicates that are True in $\boldsymbol{s}$. Likewise, we define an abstract trajectory $\bar{\boldsymbol{\tau}} = \psi(\boldsymbol{\tau}, \mathcal{P})$ as the trajectory obtained when abstracting

*mail@leon-keller.com
[1] Intelligent Autonomous Systems, TU Darmstadt, Germany
[2] German Research Center for AI, Germany
[3] Hessian Centre for Artificial Intelligence, Germany
HHonda Research Institute EU, Germany

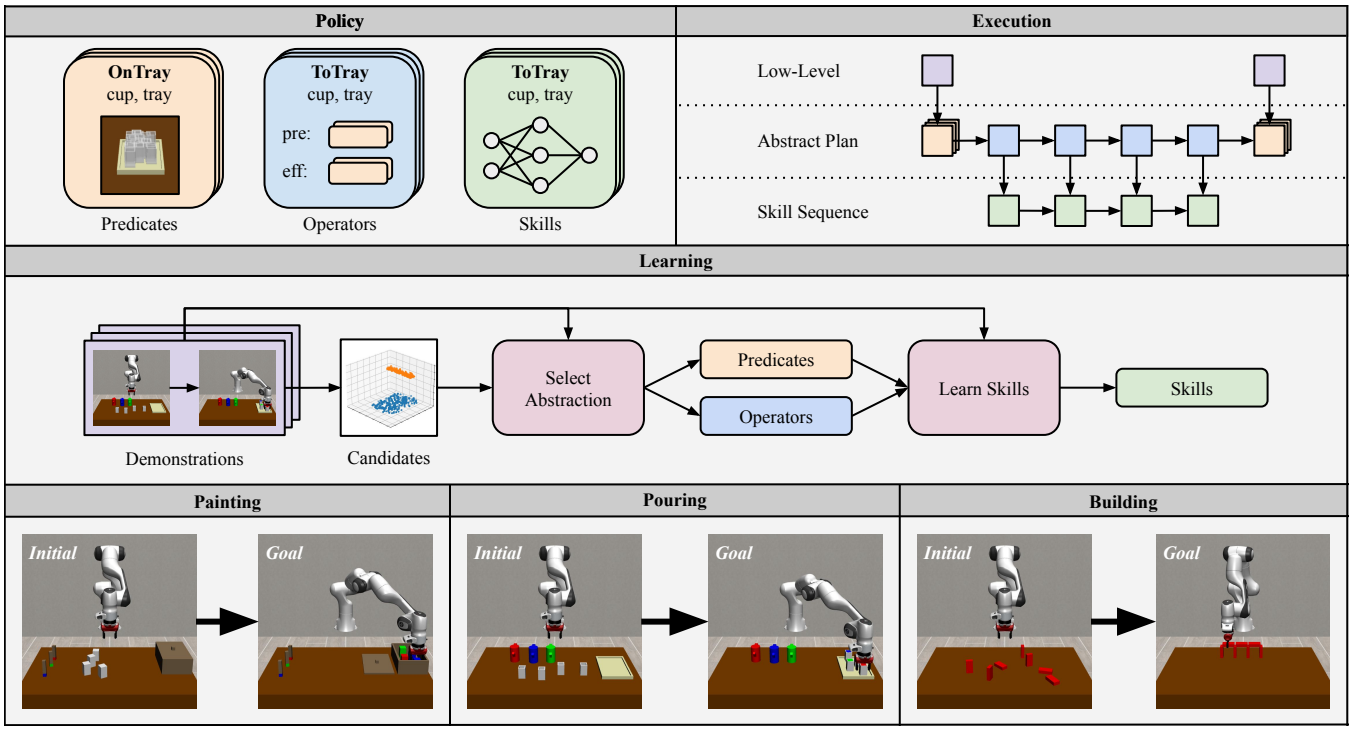

Fig. 1: *(Top Left)* The components of the neuro-symbolic policy. Predicates abstract the state-space, operators define an abstract transition model, and skills execute abstract plans. *(Top Right)* Illustration of the policy execution. First, the start and goal state are abstracted using the predicates. Following, an abstract plan is computed using the operators and planning algorithms. Lastly, the corresponding skill sequence is executed. *(Middle)* Overview of the learning pipeline. First, a set of candidate abstractions is generated based on the demonstrations. Subsequently, a subset of these candidates is selected using a novel objective function. Lastly, the learned symbolic representation is utilized to learn a set of skills with behavior cloning. *(Bottom)* The evaluation tasks. In each task, a panda robot has to manipulate objects placed on a table.

every state in $\boldsymbol{\tau}$. Grounded predicates typically do not change their truth value at every time step and, thus, $|\bar{\boldsymbol{\tau}}| \ll |\boldsymbol{\tau}|$.

2) Operators: An operator $\sigma \in \Sigma$ is characterized by a set of typed parameters Θ which specify the types of objects the operator can act on, positive and negative precondition sets $\text{pre}_+,\ \text{pre}_- \subset \mathcal{P}$, as well as positive and negative effect sets $\text{eff}_+,\ \text{eff}_- \subset \mathcal{P}$. A grounded operator $\hat{\sigma}$ is a binding of specific objects to the typed parameters of an operator. This process grounds all the predicates in the operator's precondition and effect sets. A grounded operator is applicable in $\bar{\boldsymbol{s}}$, if $\forall \hat{p} \in \text{pre}_+ : \hat{p} \in \bar{\boldsymbol{s}} \wedge \forall \hat{p} \in \text{pre}_- : \hat{p} \notin \bar{\boldsymbol{s}}$. Its execution generates the next abstract state by changing all grounded predicates in $\bar{\boldsymbol{s}}$ according to its effect sets.

3) Skills: The responsibility of a skill $\pi^i \in \Pi$ is to execute the operator it is linked to by interacting with the environment. Specifically, when a skill π^i is executed in a state $s \in \mathcal{S}$, where the corresponding operator σ^i is applicable, the skill's role is to transition the environment into a state in which the operator's effects are satisfied. As the symbolic representation abstracts the state space, many different states $s \in \mathcal{S}$ can satisfy the effects of a given operator. For example, consider an operator whose effect is defined by a predicate that measures whether a cup is placed on a tray. The corresponding skill should not be restricted to placing the cup at a single fixed position on the tray. Instead, the skill should be able to place the cup at multiple different positions, depending on the concrete circumstances. Thus, a skill $\pi^i = (f^i, g^i) \in \Pi$ in our framework consists of two components: a subgoal sampler g^i and a subgoal-conditioned controller f^i. The subgoal sampler g^i is responsible for proposing environment states that fulfill the operator's effects, while the subgoal-conditioned controller f^i is responsible for transitioning the environment toward the subgoal proposed by the sampler g^i by iteratively executing actions $a \in \mathcal{A}$ in the environment.

A. Executing a Neuro-Symbolic Policy

To solve a task with the neuro-symbolic policy, we first generate an abstract plan using the predicates $\mathcal{P}$ and operators Σ, and then execute the abstract plan in the environment using the skills Π.

1) Generate Abstract Plan: Given an initial state $\boldsymbol{s}_0 \in \mathcal{S}$ and a set of goal states $\mathcal{S}_g \subset \mathcal{S}$, we first compute an abstract initial and goal state. The abstract initial state is given by abstracting $\boldsymbol{s}_0$ using the predicates $\bar{\boldsymbol{s}}_0 = \psi(\boldsymbol{s}_0, \mathcal{P})$. Moreover, the abstract goal state $\bar{\boldsymbol{s}}_g$ is given by the set of all grounded predicates that hold true across all $\boldsymbol{s}_g \in \mathcal{S}_g$. Next, we seek a sequence of grounded operators $(\hat{\sigma}_0^i, \hat{\sigma}_1^j, \ldots)$ that induce a symbolic trajectory from $\bar{\boldsymbol{s}}_0$ to $\bar{\boldsymbol{s}}_g$, referred to as an abstract plan. Since the symbolic components are

represented in PDDL, we can employ off-the-shelf symbolic planning algorithms [49]–[51] to generate abstract plans. Specifically, we use a top-k planner to generate the best K abstract plans, assuming unit cost for each operator. Since the symbolic components abstract the environment's state and action space, not each of these abstract plans is guaranteed to be downwards refineable [12], [52]. Thus, among the generated plans, we select the one most similar to the abstract plans observed in the demonstrations. To do this, we compute the minimum Levenshtein distance [53], [54] between each generated abstract plan and the demonstrated plans, choosing the one with the smallest minimum distance.

2) Execute Abstract Plan: Once an abstract plan is selected, the next step is to sequentially execute the corresponding skill for each grounded operator in the plan. The execution of a skill π^i involves two steps: First, the skill's subgoal sampler proposes a subgoal consistent with the effects of the corresponding grounded operator $\hat{\sigma}^i$. Following, the skill's subgoal-conditioned controller is used to transition the environment towards the desired environment state. The controller continues executing actions until the effects of the corresponding grounded operator are satisfied. Once the operator's effects are achieved, the process moves on to the next grounded operator and corresponding skill in the plan.

III. Learning Neuro-Symbolic Policies

Our main contribution is twofold: First, we propose a novel method for learning predicates from raw task demonstrations. Second, to the best of our knowledge, our approach is the first to combine predicate, operator, and skill learning into a single, unified framework.

Our approaches to operator and skill learning are similar to previous work, and we detail them in Appendix E and Appendix F. In the following, we detail the main contribution of this work: predicate learning.

A. Predicate Learning

We learn predicates in a two-stage process. First, we generate a set of candidate predicates $\mathcal{C}$ by clustering absolute and relative features observed in the demonstrations. Then, we select a subset $\mathcal{P} \subset \mathcal{C}$ from these candidate predicates that trades off between a fine-grained segmentation and the complexity of the induced operator set.

1) Generate Candidates: Since the predicates should capture relations between objects, we generate candidate predicates based on relative features observed in the demonstrations. For each pair of object types $t_1, t_2 \in \mathcal{T}$ and each shared feature ξ between them, we construct a relative feature dataset $D^{\xi}_{t_1,t_2}$.

For example, consider the object types $t_1 = \text{cup}$ and $t_2 = \text{tray}$, and the feature $\xi = \text{pos}$ representing the objects' cartesian positions. To populate the relative feature dataset $D^{\text{pos}}_{\text{cup,tray}}$, we iterate over all environment states $s_t \in \tau, \tau \in \mathcal{D}$, and compute the relative position $\Delta\text{pos}_t(o_1, o_2) = \text{pos}_t(o_1) - \text{pos}_t(o_2)$ for all object pairs $o_1 \in O(\text{cup}),\ o_2 \in O(\text{tray})$. For non-position-based features, other appropriate difference functions are used. For example, to populate relative orientation datasets, we compute relative quaternions. Each relative feature is added to the dataset only if it remains constant between s_t and the consecutive state s_{t+1}. This ensures that the predicates capture stationary relations rather than transient ones during movement.

Once a relative feature dataset $D^{\xi}_{t_1,t_2}$ is constructed, we apply agglomerative clustering to identify dense regions in the feature space. Initially, each relative feature in the dataset is treated as its own cluster. We then iteratively merge the two closest clusters until no two clusters are closer than the hyperparameter ϵ. Fig. 1 shows the final clusters for the relative feature dataset $D^{\text{pos}}_{\text{cup,tray}}$. Following, a candidate predicate is created for each cluster. The typed parameters of the candidate predicate are defined as $\Theta = (t_1, t_2)$. For objects $o_1 \in O(t_1),\ o_2 \in O(t_2)$ and environment state $s_t \in \mathcal{S}$, the grounded instance of the predicate evaluates to True if the minimum distance between $\Delta\xi_t(o_1, o_2)$ and the cluster is smaller than the threshold ϵ. Following the same procedure, we generate unary candidate predicates by constructing and clustering datasets that contain absolute feature values observed in the demonstrations.

2) Select Abstraction: Next, we select a subset $\mathcal{P} \subset \mathcal{C}$ of these candidates as the final predicates set $\mathcal{P}$ and learn the corresponding operators $\Sigma(\mathcal{P}, \mathcal{D})$.

Our primary objective is to learn a symbolic representation that facilitates efficient skill learning. A predicate set segments a trajectory into subtasks by inducing a segment for each subsequence of a trajectory where the abstract state remains constant. This segmentation is critical because it defines the skills that will be learned: The shorter the segments are, the less complex the behavior is that the corresponding skill needs to learn. Therefore, we want the segmentation to be as fine-grained as possible, meaning that we aim to maximize the number of abstract states induced by the predicates. However, simply maximizing $\sum_{\tau \in \mathcal{D}} |\psi(\mathcal{P}, \tau)|$ will lead to selecting predicates that are overly specific to individual demonstrations, which would result in a large number of operators. This can be detrimental to the generalization capabilities of the symbolic planning domain, as overly specific operators are less likely to generalize to new tasks. Moreover, a large number of operators would also require a large number of skills. This is detrimental to skill learning, as a large number of skills means that each skill has access to less demonstration data. Thus, we regularize the objective by subtracting the number of operators $|\Sigma(\mathcal{P}, \mathcal{D})|$ induced by the predicate set.

Furthermore, the learned symbolic representation should generate sensible abstract plans. We assume that the abstract plans in the demonstrations are optimal. Thus, plans generated within the symbolic representation should not be shorter than the corresponding symbolic plans extracted from the demonstrations. If they were, they would be not downwards refineable and would likely miss an important predicate. Therefore, we add a constraint that ensures that the abstract plans in the demonstrations are optimal plans in the symbolic

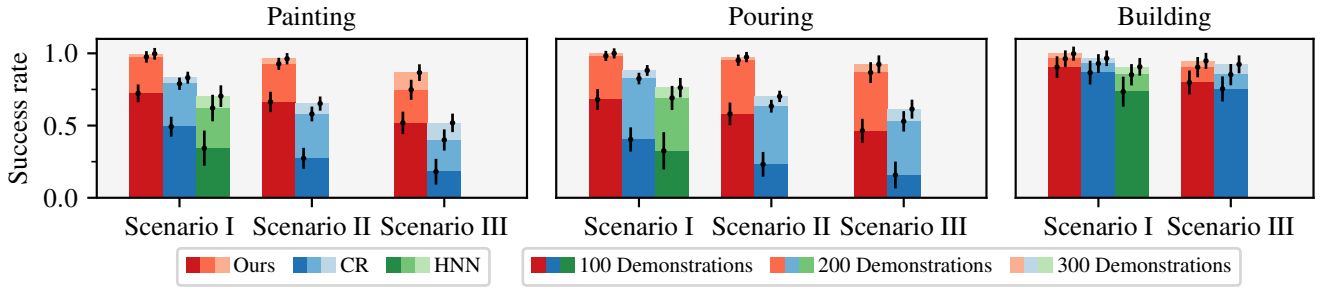

Fig. 2: Comparison between our approach and baselines in three robotic environments. The x-axis denotes the different generalization scenarios; the y-axis denotes the success rate. Results are averaged over 10 random seeds, with bars representing the mean success rate and black lines indicating the standard deviation. Color shades denote the number of demonstrations used during training.

representation. Together, we get the optimization problem

$$\max_{\mathcal{P}\subset\mathcal{C}} \sum_{\boldsymbol{\tau}\in\mathcal{D}} |\psi(\mathcal{P},\boldsymbol{\tau})| - \alpha|\Sigma(\mathcal{P},\mathcal{D})| \tag{1}$$
$$\text{s.t. } |\psi(\mathcal{P},\boldsymbol{\tau})| = |\text{plan}(\mathcal{P},\Sigma,\boldsymbol{\tau}_0,\boldsymbol{\tau}_N)|\ \forall \boldsymbol{\tau}\in\mathcal{D} \quad ,$$

where α controls the trade-off between a fine-grained segmentation and the complexity of the induced operator set.

We optimize (1) with a beam search: Initially, the beam contains only the empty predicate set. In every iteration, we evaluate all predicate sets that can be created by adding a predicate $p \in C$ to one of the predicate sets in the beam and keep only the top B subsets with the highest scores, where B is the beam width. This iterative process continues until no further improvement can be made. While this beam search does not guarantee an optimal solution, we find that it consistently produces strong results for our objective. Lastly, we select the highest scoring predicate set $\mathcal{P}$ and corresponding operator set $\Sigma(\mathcal{P},\mathcal{D})$ of the beam that fulfills the constraint.

IV. Experiments

We conduct experiments to evaluate the following questions: *(Q1)* Can our approach learn neuro-symbolic policies data-efficiently? *(Q2)* Do the learned neuro-symbolic policies generalize to tasks with unseen goals and unseen number of objects? *(Q3)* Are the learned symbols interpretable? For that, we evaluate our approach in three simulated environments (Appendix C) and compare it against two baselines (Appendix D). Moreover, we evaluate across three distinct generalization scenarios: *Scenario I* introduces initial object poses not seen during training. *Scenario II* additionally introduces unseen goals. These unseen goals include new color combinations for the `Painting` environment and new tea combinations for the `Pouring` environment. Lastly, *Scenario III* introduces more objects than during training.

Fig. 2 shows the average success rates across differing numbers of demonstrations. With 300 training demonstrations, our method achieves a high success rate across all environments and generalization scenarios. Furthermore, it outperforms both baselines across all number of demonstrations, showcasing its data-efficiency *(Q1)*. While *(HNN)* fails to generalize to tasks in *Scenario II* and *Scenario III*, our approach consistently maintains a high success rate *(Q2)*. This result highlights a major advantage of the neuro-symbolic approach: Through the learned symbols, we can benefit from the generalization capabilities of symbolic planning.

The second baseline *(CR)* learns more complex symbolic representations than our approach across all three tasks. It learns more predicates and operators, and the learned operators, on average, have more parameters. This increased complexity leads to poorer generalization of the symbolic representation to tasks in *Scenario II* and *Scenario III*. Furthermore, the larger number of operators requires a corresponding increase in the number of skills, and the additional parameters per operator complicate the state space that each skill must handle. These factors collectively contribute to *(CR)*'s lower performance relative to our method.

To address *(Q3)*, we visualize each learned predicate by overlaying images of states in which the predicate is true. These visualizations allow us to assign meaningful names to all predicates, making them easier to interpret. Fig. 3 showcases the visualization of three learned predicates. In the example shown, the predicate could be named $\text{OnTray}(\text{cup}, \text{tray})$. Once predicates are named, we can interpret the preconditions and effects of each operator and assign meaningful names to them. Fig. 4 illustrates two learned operators. With all symbols named, the abstract plans generated by the policy become fully interpretable.

V. Conclusions

In this work, we introduced a neuro-symbolic imitation learning framework that learns a symbolic planning domain and neural skills from task demonstrations. The conducted experiments show that, compared to baselines, the resulting neuro-symbolic policies offer greater data efficiency and improved generalization. Moreover, the learned symbols are interpretable. We hypothesize that the neuro-symbolic approach offers even more benefits in multi-task settings and, consequently, in future work aim to apply it in multi-task imitation learning. Moreover, we aim to explore online adaptation of both the skills and the symbolic representation to enable generalization to new object categories and failure recovery.

REFERENCES

[1] B. Zheng, S. Verma, J. Zhou, I. W. Tsang, and F. Chen, "Imitation learning: Progress, taxonomies and challenges," *IEEE Transactions on Neural Networks and Learning Systems*, 2022.

[2] B. Fang, S. Jia, D. Guo, M. Xu, S. Wen, and F. Sun, "Survey of imitation learning for robotic manipulation," *International Journal of Intelligent Robotics and Applications*, vol. 3, pp. 362–369, 2019.

[3] M. Zare, P. M. Kebria, A. Khosravi, and S. Nahavandi, "A survey of imitation learning: Algorithms, recent developments, and challenges," *IEEE Transactions on Cybernetics*, 2024.

[4] T. Osa, J. Pajarinen, G. Neumann, J. A. Bagnell, P. Abbeel, J. Peters, *et al.*, "An algorithmic perspective on imitation learning," *Foundations and Trends® in Robotics*, vol. 7, no. 1-2, pp. 1–179, 2018.

[5] J. Peters, K. Mülling, J. Kober, D. Nguyen-Tuong, and O. Krömer, "Towards motor skill learning for robotics," in *Robotics Research: The 14th International Symposium ISRR*. Springer, 2011, pp. 469–482.

[6] S. Manschitz, J. Kober, M. Gienger, and J. Peters, "Learning to sequence movement primitives from demonstrations," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014.

[7] K. Shiarlis, M. Wulfmeier, S. Salter, S. Whiteson, and I. Posner, "Taco: Learning task decomposition via temporal alignment for control," in *International Conference on Machine Learning*, 2018.

[8] D. Tanneberg, K. Ploeger, E. Rueckert, and J. Peters, "Skid raw: Skill discovery from raw trajectories," *IEEE robotics and automation letters*, 2021.

[9] J. B. Tenenbaum, C. Kemp, T. L. Griffiths, and N. D. Goodman, "How to grow a mind: Statistics, structure, and abstraction," *Science*, 2011.

[10] G. Konidaris, "On the necessity of abstraction," *Current opinion in behavioral sciences*, 2019.

[11] D. Tanneberg, E. Rueckert, and J. Peters, "Evolutionary training and abstraction yields algorithmic generalization of neural computers," *Nature Machine Intelligence*, 2020.

[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*, vol. 4, no. 1, pp. 265–293, 2021.

[13] H. Guo, F. Wu, Y. Qin, R. Li, K. Li, and K. Li, "Recent trends in task and motion planning for robotics: A survey," *ACM Computing Surveys*, vol. 55, no. 13s, pp. 1–36, 2023.

[14] M. Mansouri, F. Pecora, and P. Schüller, "Combining task and motion planning: Challenges and guidelines," *Frontiers in Robotics and AI*, vol. 8, p. 637888, 2021.

[15] N. Jetchev, T. Lang, and M. Toussaint, "Learning grounded relational symbols from continuous data for abstract reasoning," in *Proceedings of the 2013 ICRA Workshop on Autonomous Learning*, 2013.

[16] B. Bonet and H. Geffner, "Learning first-order symbolic representations for planning from the structure of the state space," 2020.

[17] I. D. Rodriguez, B. Bonet, J. Romero, and H. Geffner, "Learning first-order representations for planning from black-box states: New results," 2021.

[18] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "From skills to symbols: Learning symbolic representations for abstract high-level planning," *J. Artif. Int. Res.*, vol. 61, no. 1, p. 215–289, jan 2018.

[19] G. Andersen and G. Konidaris, "Active exploration for learning symbolic representations," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[20] G. Konidaris, L. P. Kaelbling, and T. Lozano-Perez, "Symbol acquisition for probabilistic high-level planning," in *AAAI Press/International Joint Conferences on Artificial Intelligence*, 2015.

[21] G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "Constructing symbolic representations for high-level planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 28, no. 1, 2014.

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

[23] ——, "Autonomous learning of object-centric abstractions for high-level planning," in *International Conference on Learning Representations*, 2021.

[24] 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*, vol. 75, pp. 709–745, nov 2022.

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

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

[27] M. Asai and A. Fukunaga, "Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary," 2017.

[28] M. Asai, "Unsupervised grounding of plannable first-order logic representation from images," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 29, 2019, pp. 583–591.

[29] A. Dittadi, F. K. Drachmann, and T. Bolander, "Planning from pixels in atari with learned symbolic representations," 2021.

[30] E. Umili, E. Antonioni, F. Riccio, R. Capobianco, D. Nardi, and G. De Giacomo, "Learning a symbolic planning domain through the interaction with continuous environments," in *ICAPS PRL Workshop*, 2021.

[31] 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*, vol. 37, no. 10, 2023, pp. 12 120–12 129.

[32] J. Loula, T. Silver, K. R. Allen, and J. Tenenbaum, "Discovering a symbolic planning language from continuous experience." in *CogSci*, 2019, p. 2193.

[33] H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling, "Learning symbolic models of stochastic domains," *Journal of Artificial Intelligence Research*, vol. 29, pp. 309–352, 2007.

[34] A. Curtis, T. Silver, J. B. Tenenbaum, T. Lozano-Pérez, and L. Kaelbling, "Discovering state and action abstractions for generalized task and motion planning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 36, no. 5, 2022, pp. 5377–5384.

[35] D. Tanneberg and M. Gienger, "Learning type-generalized actions for symbolic planning," in *International Conference on Intelligent Robots and Systems (IROS)*, 2023.

[36] K. Acharya, W. Raza, C. Dourado, A. Velasquez, and H. H. Song, "Neurosymbolic reinforcement learning and planning: A survey," *IEEE Transactions on Artificial Intelligence*, 2023.

[37] L. Guan, S. Sreedharan, and S. Kambhampati, "Leveraging approximate symbolic models for reinforcement learning via skill diversity," in *International Conference on Machine Learning*. PMLR, 2022, pp. 7949–7967.

[38] D. Lyu, F. Yang, B. Liu, and S. Gustafson, "Sdrl: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 2970–2977.

[39] H. Kokel, A. Manoharan, S. Natarajan, B. Ravindran, and P. Tadepalli, "Reprel: Integrating relational planning and reinforcement learning for effective abstraction," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 31, 2021, pp. 533–541.

[40] F. Yang, D. Lyu, B. Liu, and S. Gustafson, "Peorl: Integrating symbolic planning and hierarchical reinforcement learning for robust decision-making," *arXiv preprint arXiv:1804.07779*, 2018.

[41] H. Hankui Zhuo, S. Deng, M. Jin, Z. Ma, K. Jin, C. Chen, and C. Yu, "Creativity of ai: Hierarchical planning model learning for facilitating deep reinforcement learning," *arXiv e-prints*, pp. arXiv–2112, 2021.

[42] L. Illanes, X. Yan, R. T. Icarte, and S. A. McIlraith, "Symbolic plans as high-level instructions for reinforcement learning," in *Proceedings of the international conference on automated planning and scheduling*, vol. 30, 2020, pp. 540–550.

[43] M. Grounds and D. Kudenko, "Combining reinforcement learning with symbolic planning," in *European Symposium on Adaptive Agents and Multi-Agent Systems*. Springer, 2005, pp. 75–86.

[44] S. Cheng and D. Xu, "League: Guided skill learning and abstraction for long-horizon manipulation," *IEEE Robotics and Automation Letters*, 2023.

[45] T. Silver, A. Athalye, J. B. Tenenbaum, T. Lozano-Pérez, and L. P. Kaelbling, "Learning neuro-symbolic skills for bilevel planning," *arXiv preprint arXiv:2206.10680*, 2022.

[46] A. Mandlekar, C. R. Garrett, D. Xu, and D. Fox, "Human-in-the-loop task and motion planning for imitation learning," in *Conference on Robot Learning*. PMLR, 2023, pp. 3030–3060.

[47] M. J. McDonald and D. Hadfield-Menell, "Guided imitation of task and motion planning," in *Conference on Robot Learning*. PMLR, 2022, pp. 630–640.

[48] C. Aeronautiques, A. Howe, C. Knoblock, I. D. McDermott, A. Ram, M. Veloso, D. Weld, D. W. Sri, A. Barrett, D. Christianson, *et al.*,

"Pddl— the planning domain definition language," *Technical Report*, 1998.
[49] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.
[50] J. Lee, M. Katz, and S. Sohrabi, "On k* search for top-k planning," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 16, no. 1, 2023, pp. 38–46.
[51] S. Hasler, D. Tanneberg, and M. Gienger, "Efficient symbolic planning with views," *arXiv*, 2024.
[52] F. Bacchus and Q. Yang, "Downward refinement and the efficiency of hierarchical problem solving," *Artificial Intelligence*, vol. 71, no. 1, pp. 43–100, 1994.
[53] V. Levenshein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet Physics-Doklady*, vol. 10, no. 8, 1966.
[54] F. P. Miller, A. F. Vandome, and J. McBrewster, "Levenshtein distance," 2009.
[55] N. Shah, J. Nagpal, P. Verma, and S. Srivastava, "From reals to logic and back: Inventing symbolic vocabularies, actions and models for planning from raw data," *arXiv preprint arXiv:2402.11871*, 2024.
[56] R. Chitnis, T. Silver, J. B. Tenenbaum, T. Lozano-Perez, and L. P. Kaelbling, "Learning neuro-symbolic relational transition models for bilevel planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.
[57] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 5026–5033.
[58] Y. Zhu, J. Wong, A. Mandlekar, R. Martín-Martín, A. Joshi, S. Nasiriany, and Y. Zhu, "robosuite: A modular simulation framework and benchmark for robot learning," in *arXiv preprint arXiv:2009.12293*, 2020.
[59] X. Wang, "Learning planning operators by observation and practice," Ph.D. dissertation, Carnegie Mellon University, 1996.
[60] R. Stern and B. Juba, "Efficient, safe, and probably approximately complete learning of action models," *arXiv preprint arXiv:1705.08961*, 2017.
[61] P. Verma, S. R. Marpally, and S. Srivastava, "Discovering user-interpretable capabilities of black-box planning agents," *arXiv preprint arXiv:2107.13668*, 2021.

APPENDIX

A. Problem Setting

In this work, we explore imitation learning in fully observable, robotic environments. Each environment consists of a robot and multiple objects that the robot can manipulate. Each object $o \in \mathcal{O}$ is associated with a type $t(o) \in \mathcal{T}$ and described by several feature vectors $\boldsymbol{\xi}^i(o) \in \Xi(o)$, which collectively represent the object's state $\boldsymbol{s}_t(o)$ in the world. Different types of objects may be described by different feature vectors. For example, the state of an object o_1 of type $t(o_1) = \text{eef}$ may contain its cartesian pose and whether the gripper is opened or closed, while the state of an object o_2 of type $t(o_2) = \text{cube}$ may include the color of the cube. We define the environments state $\boldsymbol{s}_t \in \mathcal{S}$ at time step t as the concatenation of the states of all objects in the environment $\boldsymbol{s}_t = [\boldsymbol{s}_t(o_1),\ \boldsymbol{s}_t(o_2),\ \boldsymbol{s}_t(o_3),\ \ldots]$. Consequently, the dimensionality of the state space $\mathcal{S}$ depends on the number of objects in the environment and their respective types. The robot is controlled using an Operational Space Controller. An action $\boldsymbol{a} \in \mathcal{A}$ specifies an offset to the current end-effector pose, producing the target pose for the controller. Executing an action $\boldsymbol{a}_t \in \mathcal{A}$ transitions the system into a new state $\boldsymbol{s}_{t+1} \in \mathcal{S}$ following the environment's dynamics. We define a trajectory $\boldsymbol{\tau} = (\boldsymbol{s}_0,\ \boldsymbol{a}_0,\ \ldots,\ \boldsymbol{s}_N)$ as a sequence of state-action pairs obtained by applying actions in the environment. A task is represented by a set of goal states $\mathcal{S}_g \subset \mathcal{S}$ and a trajectory is considered to solve that task if $\boldsymbol{s}_N \in \mathcal{S}_g$.

During training, we assume access to a set of trajectories $\mathcal{D} = \{\boldsymbol{\tau}_0,\ \ldots,\ \boldsymbol{\tau}_M\}$ that demonstrate how to successfully solve a set of tasks in the environment. The objective is to learn a neuro-symbolic policy capable of solving tasks that differ from those encountered in the demonstrations. These generalization variations include unseen initial and goal object configurations, and tasks involving a greater number of objects than seen during training.

B. Related Work

Our work is closely related to methods that learn symbolic representations that abstract the state-action space of decision-making problems, particularly those focused on acquiring abstractions that facilitate high-level task planning.

A long line of research focuses on learning symbols given a predefined set of skills: Early work [15] defines predicates as radial basis function classifiers that are optimized to produce symbols that facilitate the learning of accurate transition and reward models for planning. In [16], [17], a symbolic representation is learned from given labeled graphs that encode the structure of the state-space, where nodes correspond to states and edges to skills. The construction of the symbolic representation is encoded as a Boolean satisfiability [16] or answer set programming [17] problem. Another body of work learns a symbolic representation based on the initiation and termination sets of option policies in a semi-Markov decision process [18]–[23]. Several works utilize neural networks with a binary bottleneck layer to learn symbols for planning [24]–[30]. After training the network, the binary latent vector is used to create symbols for planning. The learning objective of the networks is either reconstructing the state [27]–[29] or effect prediction [24]–[26], [30]. Multiple works invent a symbolic representation leveraging a grammar or concept language [31]–[34]. The grammar is used to generate candidate abstraction, from which the most suitable are selected using differing objective functions. The common objective of these methods is to learn symbols capable of generating high-level plans composed of predefined skills. In contrast, our method acquires a symbolic representation directly from low-level demonstrations and subsequently utilizes this representation to *learn* skills. Closest to our work, [55] learns relational symbols from raw demonstrations. Symbols are invented by identifying relational critical regions in the demonstrations. However, this method does not consider skill learning but utilizes motion planning to refine abstract plans.

Moreover, our approach is related to works that leverage symbolic representations for skill learning. The integration of symbolic planning and reinforcement learning has been extensively explored in the literature [36]–[44]. These approaches use symbolic abstractions to guide the learning process and enhance generalization. Particularly relevant to our work are studies that utilize symbolic abstractions in imitation learning [45]–[47], [56]. Symbolic representations are used to break down tasks into simpler subtasks, enabling the learning of distinct policies for each subtask.

Fig. 3: Visualization of learned predicates. Predicates are visualized by overlaying images of states in which the predicate is true.

```
(:action PAINT_BLOCK
    :parameters (?o1 robot
                 ?o2 brush
                 ?o3 block)
    :precondition (and
        (eef-at-brush ?o1 ?o2)
        (gripper-closed ?o1)
        (at-paint-position ?o2 ?o3)
        (not (same-color ?o2 ?o3))
    )
    :effect (and
        (same-color ?o2 ?o3)
    )
)
```

```
(:action OPEN_LID
    :parameters (?o1 robot
                 ?o2 lid
                 ?o3 box)
    :precondition (and
        (eef-at-lid ?o1 ?o2)
        (gripper-closed ?o1)
        (lid-on-box ?o2 ?o3)
        (not (lid-nexto-box ?o2 ?o3))
    )
    :effect (and
        (lid-nexto-box ?o2 ?o3)
        (not (lid-on-box ?o2 ?o3)
    )
)
```

Fig. 4: Illustration of learned operators. The operators are shown in PDDL-Syntax.

To the best of our knowledge, our neuro-symbolic framework is the first to simultaneously learn a relational symbolic abstraction capable of generating high-level plans and neural skills that refine these plans from raw demonstrations.

C. Environments

We evaluate our method across three simulated robotic environments. All environments utilize the `MuJoCo` [57] physics engine and the `robosuite` [58] simulation framework. In each environment, a panda robot is positioned in front of a table with various objects placed on it. In the `Building` environment, the robot must assemble rectangular blocks into a bridge-like structure. For that, the robot must pick and place blocks in the correct order to prevent the bridge from collapsing. In the `Pouring` environment, the robot must fill cups with tea and subsequently place the filled cups on a tray. There are three teapots, and each contains a different type of tea. In the `Painting` environment, the robot must paint blocks using a brush and subsequently place the painted blocks into a box. The lid of the box needs to be opened before cubes can be placed in it. There are three different brushes, each capable of applying a different color. The bottom row of Fig. 1 shows an exemplary initial and goal state for each environment.

D. Baselines

We compare our approach to two baselines: *Critical Region (CR)* ablates the proposed objective function. Instead of optimizing the proposed objective, we score and select predicates based on the notion of criticality as defined in [55]. *Hierarchical Neural Network (HNN)* ablates symbolic planning by replacing it with a neural high-level policy. Specifically, we train a neural network that takes the current and goal state as input and predicts which skill to execute.

E. Operator Learning

Here, we detail how operators Σ are learned for a given set of predicates $\mathcal{P}$. In the remainder of this work, we denote the operators induced by predicates $\mathcal{P}$ and demonstrations $\mathcal{D}$ as $\Sigma(\mathcal{P}, \mathcal{D})$.

To learn operators, we leverage symbolic state transitions observed in the demonstrations $\mathcal{D}$. Specifically, for each pair of consecutive symbolic states $\bar{\boldsymbol{s}}_t, \bar{\boldsymbol{s}}_{t+1} \in \psi(\boldsymbol{\tau}, \mathcal{P}), \boldsymbol{\tau} \in \mathcal{D}$, we construct a tuple $z = (\mathrm{pre}_+, \mathrm{pre}_-, \mathrm{eff}_+, \mathrm{eff}_-)$. In this tuple, pre_+ contains all the grounded predicates that are true in $\bar{\boldsymbol{s}}_t$, while pre_- contains those that are false. Similarly, eff_+ contains all the grounded predicates that become true when transitioning from $\bar{\boldsymbol{s}}_t$ to $\bar{\boldsymbol{s}}_{t+1}$, while eff_- contains those that become false. We then introduce a typed parameter θ for each object involved in the effect sets of the tuple and replace the objects in all the grounded predicates with the corresponding typed parameter. Objects that are not part of the effect sets remain unchanged and are treated as constants. Thus, after introducing these parameters, each tuple consists of a set of typed parameters Θ and lifted precondition and effect sets.

Following, we group the tuples $z \in Z$ based on their lifted effect sets. Specifically, two tuples $z_i, z_j \in Z$ belong to the same group G_k if and only if their effect sets and typed parameters are identical. We then construct an operator σ_k for every group G_k: The operators typed parameters $\Theta(\sigma_k)$ and effect sets $\mathrm{eff}_+(\sigma_k)$, $\mathrm{eff}_-(\sigma_k)$ are defined by the groups typed parameters and effect sets. The operator's precondition sets are given by the intersection of the corresponding precondition sets of all tuples in the group: $\mathrm{pre}_+(\sigma_k) = \cap_{z_i \in G_k} \mathrm{pre}_+(z_i)$, $\mathrm{pre}_-(\sigma_k) = \cap_{z_i \in G_k} \mathrm{pre}_-(z_i)$. This assumes that for every lifted effect set in the demonstrations, there is exactly one lifted precondition set. Similar approaches for operator learning are widely used in the literature [31], [35], [45], [55], [56], [59]–[61].

F. Skill Learning

Our approach to skill learning is similar to previous work [45]. Given predicates $\mathcal{P}$ and operators Σ, we learn a subgoal-conditioned controller f_i and a subgoal sampler g_i for each operator $\sigma_i \in \Sigma$. In contrast, we transform the states differently and adopt non-parametric distributions to represent the samplers.

1) Generate Skill Datasets: We decompose the demonstrations into distinct skill datasets by segmenting each $\boldsymbol{\tau} \in \mathcal{D}$ according to the abstraction defined by the predicates $\mathcal{P}$ and operators Σ. A segment (j, k) is induced by each subsequence $(\boldsymbol{s}_j, \boldsymbol{a}_j, \ldots, \boldsymbol{s}_k, \boldsymbol{a}_k)$ of a trajectory, during which the abstract state remains constant. We define the state that immediately follows the segment $\boldsymbol{s}_{k+1}$ as the segments' goal. We determine to which skill dataset each segment (i, j) should be added by examining which operator's σ preconditions and effects match to the transition from the abstract state of the segment $\psi(\boldsymbol{s}_{(j:k)}, \mathcal{P})$ to the abstract state of the segment's goal $\psi(\boldsymbol{s}_{k+1}, \mathcal{P})$. Before adding the segment to the skill dataset, we transform the states to only contain

information relevant to the execution of operator σ. For that, we define a transformation function ϕ_σ, which maps a state $\boldsymbol{s}$ to a transformed state based on the predicates in the effect sets of the operator. For instance, if the effect sets only contain the predicate $\mathrm{OnTop}(o_1, o_2) = \mathrm{True}$, then $\phi_\sigma(\boldsymbol{s})$ will only return the relative pose between o_1 and o_2. We then add the transformed segment to the skill dataset corresponding to operator σ: $D_\sigma = D_\sigma \cup \{(\phi_\sigma(\boldsymbol{s}_l),\ \boldsymbol{a}_l,\ \phi_\sigma(\boldsymbol{s}_{k+1}))\}_{l=j:k}$.

2) Learn Skills: After the datasets are constructed, we learn a skill $\pi^i = (f^i, g^i)$ for each dataset D_{σ^i}. Learning a goal-conditioned controller f^i reduces to a supervised learning problem. For each f^i, we train a multi-layer perception using behavior cloning. Learning a goal sampler g^i is framed as a probability density estimation problem. For each g^i, we model the distribution of possible goals using kernel density estimation with radial basis function kernels.