Modular Hierarchical Reinforcement Learning for Robotics: Improving Scalability and Generalizability

Mihai Anca [1] **Mark Hansen** [2] **Matthew Studley** [2]

Abstract

We present a novel software architecture for reinforcement learning applied to robotics that emphasizes modularity and reusability. Our method treats each agent as a plug-and-play ROS node that can be easily integrated into a larger HRL system, similar to using software libraries in programming. This modular approach improves the scalability and generalizability of pre-trained reinforcement learning agents. We demonstrate the effectiveness of our method by solving the real-world task of stacking three objects with two different robots that were trained only in simulation. Our results show that the modular approach significantly reduces the training and setup time required compared to a vanilla reinforcement learning baseline. Overall, our work showcases the potential of using trained agents as modules to enable the development of more complex and adaptable robotics applications.

1. Introduction

The concept of software libraries, similar to using modules or plug-and-play components, has been used in programming since the early days of computer programming. Libraries allow developers to reuse code and reduce the amount of time and effort required to write new code from scratch. Moreover, the introduction of package managers has seen increases in productivity and the development of more complex software systems, on top of reducing the entry level for new software developers. Unfortunately, that is not the case for reinforcement learning. There are many frameworks for training reinforcement learning agents ((Makoviichuk & Makoviychuk, 2022; Raffin et al., 2021; Liang et al., 2017; François-Lavet, 2016)), however, there are no widely accepted solutions that allow agents to be treated as modular components that can be combined into larger systems. This makes it difficult to reuse pre-trained agents across different tasks or to scale up the size of the system.

To address this issue, we propose a new software architecture based on Hierarchical Reinforcement Learning (HRL) and Robotics Operating System (ROS) (Quigley et al., 2009) that alleviates the Sim2Real burden of previous RL methods. Our approach is modular, incorporating the use of pre-trained agents as plug-and-play components. Moreover, it leverages the hierarchy of tasks that is common in robotics to create a system of modules that can be easily combined to solve more complex problems. By treating each RL agent as a ROS node, we are able to reuse pre-trained agents across different tasks and integrate them into larger systems with ease. This allows for a more scalable and generalizable approach to reinforcement learning, which has traditionally been limited by its lack of modularity.

Contrary to RL, robotic systems such as autonomous vehicles (Munir et al., 2018) have long made use of modular designs thanks to the ROS framework, which has established itself as a leading open-source framework for building robotic systems. Its suite of tools, libraries, and conventions provides a foundation for the creation of complex robotic applications. ROS's key strength is its ability to facilitate communication between different components, making it an ideal ecosystem for the creation and sharing of a library of pre-trained reinforcement learning agents.

The field of robotics includes both emerging technologies and established systems. For example, humanoid assistant robots are still in development and not yet available on the market, while assembly-line robots have undergone numerous improvements over the years. One way to accelerate the progress of assistant robot research is by leveraging the primitive skills of picking, placing, and manipulating that have already been developed for assembly line robots. In turn, the experiences gathered by interacting with more fragile items and people in household environments could be used to further upgrade assembly line robots. By sharing knowledge and skills between these two areas of robotics,

[1]University of Bristol, Bristol, United Kingdom [2]University of the West of England, Bristol, United Kingdom. Correspondence to: Mihai Anca <mihai.anca@bristol.ac.uk>.

Workshop on New Frontiers in Learning, Control, and Dynamical Systems at the International Conference on Machine Learning (ICML), Honolulu, Hawaii, USA, 2023. Copyright 2023 by the author(s).

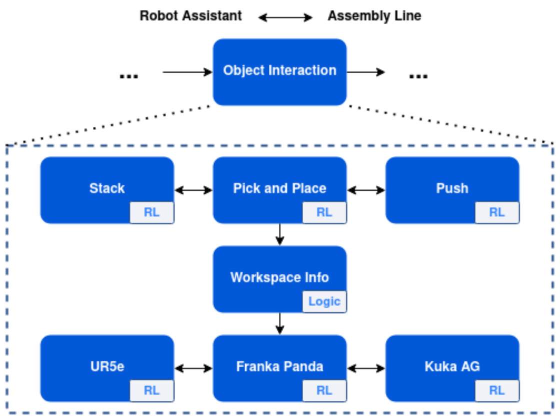


Figure 1. Shared Object Interaction module between a robot assistant solution and an assembly line system. The module comprises three layers with interchangeable parts depending on the specifications of the task and the robotic system used.

we can create a more efficient and effective ecosystem for robot development and innovation (see Fig 1 for a visual example).

Although there are efforts to create large foundation models (Ahn et al., 2022) that can, in theory, replace the described work through fine-tuning, they often require significant resources and infrastructure, making them challenging to use. Additionally, they depend on the full release by publishers.

In the following sections, we describe the method in more detail and demonstrate its effectiveness on the task of stacking multiple objects using two separate robotic arms in the real world. Based on the example given earlier, the task chosen and the resulting system can be seen as a small module in itself, which can be integrated as part of a robot assistant or as part of an assembly line. By comparing with a vanilla RL architecture such as (Anca et al., 2023), we demonstrate that our approach significantly speeds up the setup and training time, while also providing a structure that benefits scalability.

2. Background

The software architecture described in this paper is mainly inspired by previous hierarchical reinforcement learning frameworks such as the Options ((Sutton et al., 1999; Stolle & Precup, 2002)) and Feudal (Dayan & Hinton, 1992) frameworks. A direct comparison between these and our architecture is presented in Section 3.

Currently, there is a lack of standardization in adapting previous work to new tasks or combining multiple pre-trained agents into more complex systems. Despite this, RL has achieved great success in the area of robotics and abstract environments, such as games. For instance, research in complex games such as Starcraft (Vinyals et al., 2019), Chess (Silver et al., 2017) and Go (Silver et al., 2016) have achieved great results in abstract worlds, and recent work in robotics has demonstrated success in complex manipulation tasks such as in-hand manipulation (Andrychowicz et al., 2020), painting (Park et al., 2022), and speed folding (Avigal et al., 2022).

A potential solution is presented in (Andreas et al., 2017), where the authors introduce MMRL, a modular approach for achieving efficient and effective multitask reinforcement learning. MMRL introduces the concept of policy sketches, which are modular building blocks that can be combined to create policies for different tasks. MMRL's main strength is its ability to learn multiple tasks concurrently, which can lead to significant time savings, compared to learning each task separately. However, MMRL's performance is limited by the accuracy of the policy sketches used, which can be difficult to design and fine-tune. Moreover, the authors note that the framework still requires significant manual engineering and is not yet scalable to more complex domains.

A different method is presented in (Haarnoja et al., 2018). In this paper, the authors propose a novel approach that combines reinforcement learning with a compositional architecture for robotic manipulation. The authors demonstrate that their approach can effectively learn complex manipulation tasks with minimal prior knowledge, while also being able to generalize to unseen objects and scenarios. However, the approach cannot scale well because it relies on finding a good weighting of different policies for each task.

In the following subsections, we will explore the key theoretical concepts essential for this discussion.

2.1. Reinforcement Learning

Reinforcement learning is a machine learning technique that trains an agent to make decisions based on feedback from its environment, with the aim of maximizing a reward signal defined by the system designer. In this technique, the agent interacts with the environment by taking actions and receiving feedback in the form of rewards. The goal of the agent is to learn a policy that maps states to actions that maximizes the expected cumulative reward over time. This is usually formulated as a Markov Decision Process (MDP), a mathematical framework for modeling decision-making problems while dealing with uncertainty.

The expected cumulative reward over time is represented by the expected value function, denoted as $V(s)$, which represents the expected reward starting from a state s and following a particular policy. The optimal policy is the one that maximizes the expected value function, i.e. $\pi^*(s) = arg\,max\ V(s)$. The value function can be estimated using the Bellman equation, which expresses the value function of a state in terms of the values of its neighbor states:

$$V(s) = \mathbb{E}\left[R(s,a) + \gamma V(s')\right],$$

where $R(s,a)$ is the expected reward of taking action a in state s, s' is the next state, γ is the discount factor that trades off immediate and future rewards.

Deep Q-Networks (DQN) (Mnih et al., 2015) use a reinforcement learning algorithm that combines Q-learning with deep neural networks to handle high-dimensional state spaces. DQN uses a replay buffer to store experience tuples, which are used to update the Q-network in batches to improve sample efficiency and stability. DQN is a good choice when the state space is high-dimensional and discrete actions are sufficient to solve the problem.

Proximal Policy Optimization (PPO) (Schulman et al., 2017) is another popular reinforcement learning algorithm that uses a trust region optimization approach to update the policy. PPO is designed to balance exploration and exploitation by constraining the size of policy updates, which improves stability and convergence. PPO has been shown to perform well on a variety of tasks, including continuous control (Rudin et al., 2022) and robotic manipulation (Andrychowicz et al., 2020).

2.2. Hierarchical Reinforcement Learning

Hierarchical reinforcement learning (HRL) is a subfield of reinforcement learning. It aims to improve the efficiency and scalability of learning by breaking down a task into sub-tasks with different levels of temporal and/or spatial abstractions. The main idea is to learn a hierarchy of policies, each of which controls a different level of abstraction, such that the policies at the lower levels can be reused for multiple higher-level tasks, leading to faster learning and improved generalization.

One of the earliest and most influential HRL frameworks is the Options architecture (Sutton et al., 1999; Stolle & Precup, 2002). It introduces "options" as temporally extended actions that are defined as Markov decision processes (MDPs) with their own policies and termination conditions. The follow-up work of (Bacon et al., 2017) improves the Options framework by formalizing it in the context of neural networks. This approach can significantly reduce problem complexity and increase the agent's flexibility by enabling it to select between different options. One limitation is the need to train many low-level agents for each option, without the possibility of sharing pre-trained ones due to the manager having to re-learn the capabilities of each low-level agent. The Options framework has been applied successfully to robotic control (Krishnan et al., 2017) and game-playing (Zhang & Whiteson, 2019).

Another HRL framework is Feudal Reinforcement Learning (Dayan & Hinton, 1992). It is a structured approach to control, where a decision-making agent, called the manager, selects goals for sub-managers responsible for achieving those goals. One of the significant advantages of feudal learning is that it allows sub-managers to choose their subgoals that align with the assigned main goal. Thanks to the reward and information-hiding mechanisms, sub-managers can learn to achieve sub-goals even if the manager was mistaken in setting them. This approach permits decision-making at a higher level, making it easier for managers to have a broad understanding of the system's state and focus on their chosen tasks' details.

While both the Option Critic framework and Feudal RL have been successful in different domains, the Option Critic framework has been found to be more scalable and efficient due to the fact that the options can be learned independently. On the other hand, Feudal RL has been shown to be more interpretable and better suited for hierarchical tasks due to its clear separation of high-level goals and low-level skills.

2.3. Sim2Real

Sim2Real, or simulation-to-reality transfer, is a rapidly growing area of robotics research. The fundamental idea is to use simulations as a safe and cost-effective way to train machine learning models for various robotic tasks, and then transfer the learned policies to the real world. However, the differences between simulation and reality in many simulation tools have led to limited deployment of policies in reality. To reduce this gap, researchers (Dimitropoulos et al., 2022) propose more realistic simulations, as well as more robust policies.

Current approaches to obtain robust policies include the introduction of perturbations in the environment (Pinto et al., 2017), and domain randomization (Tobin et al., 2017; Ding et al., 2021; Peng et al., 2018). The former involves introducing changes in the environment, while the latter randomizes various simulation parameters during training to encourage the model to learn more generalized policies.

The proposed methods increase the complexity that the agent must incorporate into its learning, resulting in longer training times. To mitigate this effect, it is possible to separate the modules that interact directly with the environment and apply the required techniques only to them. This approach reduces the burden of complexity on higher-level agents, while still maintaining the benefits associated with increased generalizability of low-level agents.

3. Method

Our architecture is based on RL agents trained in isolation, that can later be used together to form a more complex

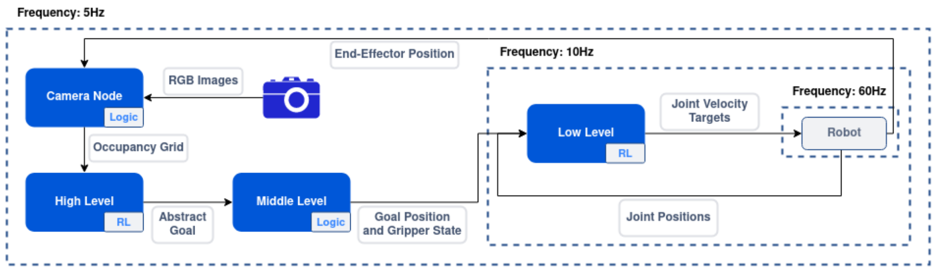


Figure 2. The experimental setup built on top of ROS consisting of two RL agents (High Level and Low Level) communicating through the use of a logic-based Middle Level. The high-level agent receives the observation from the Camera Node and generates a goal, which then gets passed to the Middle Level. Here, the abstract goal is converted to 3D coordinates using the workspace dimensions and depending on the state, the gripper is engaged. The Low Level, upon receiving the converted goal and the observation from the robot, generates per-joint velocity targets at a lower frequency than the high and middle levels. Finally, the robot controller follows the velocity commands at a yet higher frequency.

system. The integration is done using ROS, where each agent is wrapped into a node. There are two types of agents: a high-level agent, generally operating in a discrete version of the real environment that produces abstract goals; and a low-level agent, which produces actions that affect the environment directly, based on the goals provided from a linked high level.

Similarly, we have high and low layers, each of which consists of one or more pre-trained agents. The communication between these layers is facilitated by another middle layer, which consists of logic-based nodes written by engineers. Their role is to introduce task-specific knowledge into the system, such as real-world measurements/dimensions, and to convert the abstract goals into continuous ones that can be used by the low layers. An added benefit of organising everything in multiple layers is that each can run at different frequencies, allowing for available resources to be efficiently allocated. Please see Figure 2 for an example of how this was used in our experiments.

In the context of HRL, the proposed software architecture is a hybrid between the Feudal (Dayan & Hinton, 1992) and the Option-Critic (Bacon et al., 2017) frameworks. In the Option-Critic architecture, one high-level agent is trained to identify the states in which each of the provided low-level agents is most suitable. Similar to our proposed method, low-level agents are trained disjointly, however, if the availability of the low-level agents changes, the training process must be restarted. This problem has been addressed by introducing middle layers, through which engineers can control the logical flow of the system.

In the feudal framework, each level can only have one agent, with higher-level agents producing goals for lower-level agents. If we replace a low-level agent with another agent that can achieve the given goals with the same degree of accuracy, the overall performance will not be affected. We maintain this property in our architecture by making use of the same goal-setting communication style between different levels. However, having only one agent per level can prove limiting in complex systems, such as the robot assistant example discussed in the Introduction. There, low-level agents could directly influence the environment through speech, movement and manipulation, therefore, having specialised nodes for each of these tasks could prove beneficial.

4. Evaluation

Task decomposition has been proven to decrease training time in RL agents (Pateria et al., 2021; Narvekar et al., 2020). This is especially important in real-world applications, where training data is sparse, re-training is costly, and redundant modules are necessary to ensure the continual operation of the system. This section describes the experiments we used to highlight the benefits of modularity and hardware independence properties of our software architecture.

Our real-world experiments involve a robotic arm positioned in front of a table with three cubes (see Fig. 3). Each object has an attached Aruco tag that is used to translate the position of each object into an occupancy grid, using the Intel RealSense camera. To successfully complete the task, the system controlling the robot arm must stack all three cubes at a desired goal location.

We used the 7 DOF Panda robotic arm by Franka Emika, and the 6 DOF UR5e from Universal Robots, to show the generalizability of our approach to two different hardware systems. Both are controlled through the CRI (Lloyd, 2022) library using joint-based velocity control. The Panda arm uses the stock gripper, while the UR5e is equipped with the Robotiq 2F-85 gripper.

We divide the task and train our agents in two simulated environments: 1) a high-level discrete environment, designed for training an agent to plan a series of moves that would

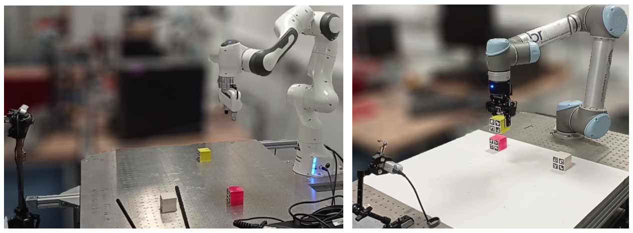
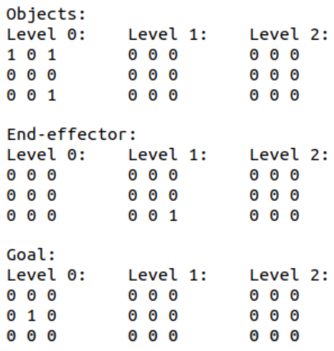


Figure 3. Franka Panda robot (left) and UR5e robot (right) positioned in front of 3 blocks. An Intel Realsense camera is used to track the Aruco tags attached to the blocks.

lead to stacking a tower of cubes; and 2) a low-level continuous environment, where a robotic arm is simulated and trained to reach specific goal positions. A logic-based middle level sits in between these two, containing task-specific knowledge, such as the dimension of the workspace and the size of objects. Using this information, the middle layer translates the abstract goals, generated by the high-level agent, into 3D world coordinates that the low-level agent can use.

4.1. High-level

Using an abstract environment to train the high-level agent can significantly reduce the training time of each agent, allowing for faster iteration through solutions. Decoupling the high-level from the low-level agent provides the first boost in training time, but the major step comes from working with a discrete world.



```
Objects:
Level 0:     Level 1:     Level 2:
1 0 1        0 0 0        0 0 0
0 0 0        0 0 0        0 0 0
0 0 1        0 0 0        0 0 0

End-effector:
Level 0:     Level 1:     Level 2:
0 0 0        0 0 0        0 0 0
0 0 0        0 0 0        0 0 0
0 0 0        0 0 1        0 0 0

Goal:
Level 0:     Level 1:     Level 2:
0 0 0        0 0 0        0 0 0
0 1 0        0 0 0        0 0 0
0 0 0        0 0 0        0 0 0
```

Figure 4. High-level environment observation formed from 3 sparse matrices containing information about object, end-effector and goal positions.

For simplicity, to prove our approach, we chose a 3x3x3 size to represent all possible object positions. The agent can grasp, release, and move the end-effector instantly between these positions. Its goal is to stack the three objects at a desired location.

The observation space (Fig. 4) consists of three sparse 3x3x3 boolean matrices, with 1s representing the objects, end-effector, and goal positions. The action space is discrete and has a size of 29, with the first 2 actions representing grasping and releasing, and the remaining 27 representing possible movements.

The reward is sparse, with an additional signal for each correctly placed object. To increase the speed of convergence even further, a negative reward has been added for releasing objects into the air.

4.2. Low-level

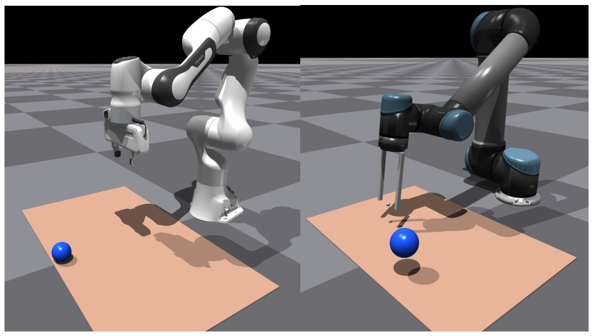


Figure 5. Low-level environment built using Nvidia Isaac Gym for training agents in controlling the Franka Panda and Universal Robots UR5e robotic arms to reach specific 3D locations.

The low-level environment (Fig. 5) comprises a Proximal Policy Optimization (PPO) agent trained on a robot arm simulated in Nvidia Isaac Gym (Makoviychuk et al., 2021) to perform the Reach task. The observation and action space is continuous, and the goal is represented by a set of 3D coordinates that the end-effector must reach within the episode's time limit. The agent learns to set velocity targets for each joint to maintain a vertical orientation of the end-effector while trying to reach the goal position.

Similar to the examples provided in (Makoviychuk et al., 2021), we incentivize the agent to reach the goal position by using a dense reward function based on the distance between the end-effector and the goal position. The reward is calculated as the inverse of the Euclidean distance, and is bounded between $[-1, 1]$ using the $\tanh$ function.

To facilitate Sim2Real transfer, we introduced domain randomization (Peng et al., 2018) as noise to observations, and slight adjustments to joint parameters such as damping, range, friction, and mass.

Building on the approach advocated in (Böhm et al., 2022), we included the time since the last observation as part of the next one. We also added noise to this value to simulate communication delays in the real world.

Consistent with (Peng et al., 2018), we found that maintaining a fixed frequency control loop even when training the

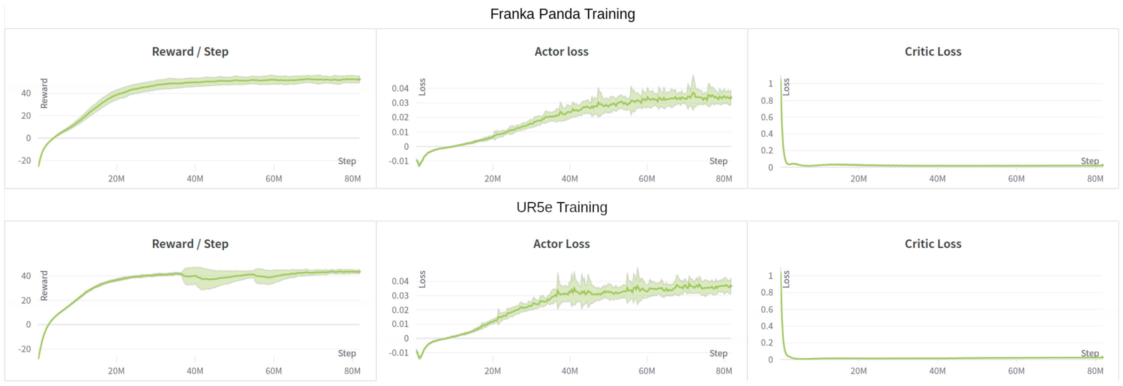


Figure 6. Low-level training graphs. The task is solved upon reaching a mean reward of 40. UR5e achieves a better score due to a reduced workspace and a lower number of joints to be controlled.

agent, has a significant positive impact on its transferability.

5. Results

Stacking items is one of the primary uses for robotic arms, therefore, we evaluate the efficacy of our proposed method on the task of stacking three blocks in the real world. By comparing the required training time of our method against a vanilla RL baseline (Anca et al., 2023), we demonstrate that decomposing the task significantly decreases training time. Our experiments also show that all trained agents can form a basis for a library of reusable behaviours. Furthermore, by employing two different robots, we prove that our system is hardware independent.

The Stable Baselines 3 framework (Raffin et al., 2021) is used to train a DQN agent for the high-level environment (see Fig. 7). For the low-level, two separate agents were trained on the Franka Panda and the UR5e robots, using the PPO implementation provided by the RL Games framework (Makoviichuk & Makoviychuk, 2022) (see Fig. 6). Hyperparameters can be found in the Appendix.

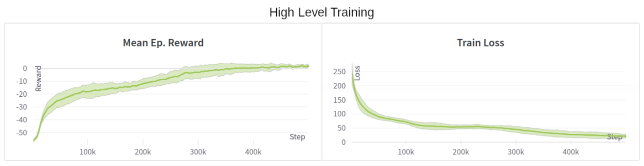


Figure 7. High-level training graphs. The task is solved upon reaching a mean reward of 0. The reward is negative to encourage the agent to take meaningful actions at each time step.

The chosen baseline (Anca et al., 2023) is a PPO agent trained using curriculum learning and reward shaping on the same cube stacking task. Their method trains directly on the task without decomposing it or applying any temporal or spatial abstraction. Table 1 shows the significant improvement in training time gained by decomposing the task across three layers. Furthermore, hyperparameter search and fine-tuning operations have become much easier with a modular approach. Additionally, the baseline method does not include domain randomization, which generally increases the complexity of the task during training. In contrast, the low-level agent trained in this work leverages domain randomization to enhance the transferability at a much lower training time cost.

Table 1. Training time comparison between our two agents (high and low level) and a vanilla RL baseline.

Name	Resources	Training time
Low-level	1 GPU - 4096 envs	24min
High-level	CPU - 1 env	2h
Baseline	2 GPUs	20h

5.1. Real world experiment

For simple tasks - such as cube stacking, the low-level controller can be replaced with an inverse kinematics (IK) logic module. The IK module directly calculates the joint angles required to reach a given goal pose. However, our experiments use the cube stacking task only to demonstrate the effectiveness of the proposed architecture. Our solution can be transferred to more complex environments where IK would fail to find a direct solution. To achieve this, we combined the high-level agent with an IK module and a trained low-level RL agent separately. We tested the combined agents on both robots in 10 randomized trials.

The results of our experiments show that the high-level agent is able to successfully complete the cube stacking task with both the IK module and the trained low-level RL agent. This demonstrates the modularity of the proposed architecture, as the high-level agent is able to work with different types of low-level controllers with no changes required to the high or middle layers.

Similarly, when repeating the experiment on the second robot, only the low-level agent controlling the robot must be swapped with the newly trained one. The middle layer, which is logic-based, undergoes minor modifications, such as updating the workspace dimensions due to a slightly reduced workspace size. However, all other modules and communication remain the same. The software architecture presented is, therefore, hardware independent.

All combinations of high and low levels (see Appendix for a list of trials) successfully stack all three cubes across all trials. Visit this *link* to watch a successful trial with each robotic arm, where both the high and low levels are controlled by RL.

6. Conclusion

We have introduced a novel software architecture designed for reinforcement learning agents in the area of robotics. It provides a way of separating task-specific RL solutions from their implementations, reducing the transfer burden to a real-world system. We have shown how this approach leads to far faster learning and hardware-independent deployment compared to a vanilla algorithm. We have proved the modularity of this approach in our experiments by solving the real-world task of stacking three cubes, using two different robotic arms. Due to no changes being required at the high and middle layers, swapping one low-level agent for another took significantly less time than building the system from scratch. The system's modular nature improves scalability and generalizability, by allowing complex tasks to be split into sub-tasks that can be more easily solved - either through an abstract environment or direct training. Our work suggests that building a library of pre-trained RL agents is now possible, and will significantly reduce the costs of applying RL on complex real-world applications.

References

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Gopalakrishnan, K., Hausman, K., Herzog, A., et al. Do as i can, not as i say: Grounding language in robotic affordances. *arXiv preprint arXiv:2204.01691*, 2022.

Anca, M., Studley, M., Hansen, M., Thomas, J. D., and Pedamonti, D. Achieving goals using reward shaping and curriculum learning. *arXiv preprint arXiv:2206.02462v2*, 2023.

Andreas, J., Klein, D., and Levine, S. Modular multitask reinforcement learning with policy sketches. In *International Conference on Machine Learning*, pp. 166–175. PMLR, 2017.

Andrychowicz, O. M., Baker, B., Chociej, M., Jozefowicz, R., McGrew, B., Pachocki, J., Petron, A., Plappert, M., Powell, G., Ray, A., et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Avigal, Y., Berscheid, L., Asfour, T., Kröger, T., and Goldberg, K. Speedfolding: Learning efficient bimanual folding of garments. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1–8. IEEE, 2022.

Bacon, P.-L., Harb, J., and Precup, D. The option-critic architecture. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31, 2017.

Böhm, P., Pounds, P., and Chapman, A. C. Non-blocking asynchronous training for reinforcement learning in real-world environments. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10927–10934. IEEE, 2022.

Dayan, P. and Hinton, G. E. Feudal reinforcement learning. *Advances in neural information processing systems*, 5, 1992.

Dimitropoulos, K., Hatzilygeroudis, I., and Chatzilygeroudis, K. A brief survey of sim2real methods for robot learning. *Advances in Service and Industrial Robotics: RAAD 2022*, pp. 133–140, 2022.

Ding, Z., Tsai, Y.-Y., Lee, W. W., and Huang, B. Sim-to-real transfer for robotic manipulation with tactile sensory. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 6778–6785. IEEE, 2021.

François-Lavet, V. Deep reinforcement learning framework, 2016. URL `https://github.com/VinF/deer`.

Haarnoja, T., Pong, V., Zhou, A., Dalal, M., Abbeel, P., and Levine, S. Composable deep reinforcement learning for robotic manipulation. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 6244–6251. IEEE, 2018.

Krishnan, S., Fox, R., Stoica, I., and Goldberg, K. Ddco: Discovery of deep continuous options for robot learning from demonstrations. In *Conference on robot learning*, pp. 418–437. PMLR, 2017.

Liang, E., Liaw, R., Moritz, P., Nishihara, R., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I., and Stoica, I. Rllib: Abstractions for distributed reinforcement learning. arxiv e-prints, page. *arXiv preprint arXiv:1712.09381*, 2017.

Lloyd, J. Common robot interface, 2022. URL `https://github.com/jlloyd237/cri`.

Makoviichuk, D. and Makoviychuk, V. Rl games, 2022. URL https://github.com/Denys88/rl_games.

Makoviychuk, V., Wawrzyniak, L., Guo, Y., Lu, M., Storey, K., Macklin, M., Hoeller, D., Rudin, N., Allshire, A., Handa, A., et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature*, 518(7540): 529–533, 2015.

Munir, F., Azam, S., Hussain, M. I., Sheri, A. M., and Jeon, M. Autonomous vehicle: The architecture aspect of self driving car. In *Proceedings of the 2018 International Conference on Sensors, Signal and Image Processing*, pp. 1–5, 2018.

Narvekar, S., Peng, B., Leonetti, M., Sinapov, J., Taylor, M. E., and Stone, P. Curriculum learning for reinforcement learning domains: A framework and survey. *J. Mach. Learn. Res.*, 21(1), jan 2020. ISSN 1532-4435.

Park, Y., Jeon, S., and Lee, T. Robot learning to paint from demonstrations. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3053–3060. IEEE, 2022.

Pateria, S., Subagdja, B., Tan, A.-h., and Quek, C. Hierarchical reinforcement learning: A comprehensive survey. *ACM Comput. Surv.*, 54(5), jun 2021. ISSN 0360-0300. doi: 10.1145/3453160. URL https://doi.org/10.1145/3453160.

Peng, X. B., Andrychowicz, M., Zaremba, W., and Abbeel, P. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 3803–3810. IEEE, 2018.

Pinto, L., Davidson, J., Sukthankar, R., and Gupta, A. Robust adversarial reinforcement learning. In *International Conference on Machine Learning*, pp. 2817–2826. PMLR, 2017.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A. Y., et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, pp. 5. Kobe, Japan, 2009.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., and Dormann, N. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Rudin, N., Hoeller, D., Reist, P., and Hutter, M. Learning to walk in minutes using massively parallel deep reinforcement learning. In Faust, A., Hsu, D., and Neumann, G. (eds.), *Proceedings of the 5th Conference on Robot Learning*, volume 164 of *Proceedings of Machine Learning Research*, pp. 91–100. PMLR, 08–11 Nov 2022. URL https://proceedings.mlr.press/v164/rudin22a.html.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Stolle, M. and Precup, D. Learning options in reinforcement learning. In *Abstraction, Reformulation, and Approximation: 5th International Symposium, SARA 2002 Kananaskis, Alberta, Canada August 2–4, 2002 Proceedings 5*, pp. 212–223. Springer, 2002.

Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2): 181–211, 1999.

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., and Abbeel, P. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pp. 23–30. IEEE, 2017.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019.

Zhang, S. and Whiteson, S. Dac: The double actor-critic architecture for learning options. *Advances in Neural Information Processing Systems*, 32, 2019.

A. Appendix

Table 2. High-level hyperparameters

Parameter	Value
Timesteps	500000
learning rate	8e-4
batch	128
learning start	1000
gamma	0.82
target update freq	30
gradient steps	199
exploration fraction	0.0258
exploration final eps	0.067

Table 3. Low-level hyperparameters

Parameter	Value
num envs	1024
ep. length	16
distReward	2.0
doneReward	7.5
actionPenalty	0.001
orietationPenalty	5
distanceThreshold	0.005
noiseSteps	True
delaySteps	True
controlFrequencyInv	6
mlp units	[128, 128, 128]
activation	elu
fixed sigma	True
gamma	0.99
tau	0.95
learning rate	5e-4
lr schedule	adaptive
horizon length	16
minibatch size	8192

Table 4. Experiment variants

Setup	Goal

1 0 0
1 0 0
0 1 0
	[1, 1]

0 1 0
0 1 0
0 0 1
	[1, 1]

0 0 0
0 0 1
1 0 1
	[0, 1]

1 0 1
0 1 0
0 0 0
	[1, 2]

1 0 0
0 0 0
1 0 1
	[0, 2]

0 0 0
1 1 1
0 0 0
	[0, 1]

0 1 1
0 1 0
0 0 0
	[2, 1]

0 1 0
0 1 0
1 0 0
	[0, 0]

0 0 0
0 1 1
0 1 0
	[2, 2]

0 1 0
1 0 1
0 0 0
	[0, 1]