DEEP MODULAR REINFORCEMENT LEARNING FOR PHYSICALLY EMBEDDED ABSTRACT REASONING

Peter Karkus[1,2], **Mehdi Mirza**[1], **Arthur Guez**[1], **Andrew Jaegle**[1], **Timothy Lillicrap**[1], **Lars Buesing**[1], **Nicolas Heess**[1], **Theophane Weber**[1]
[1]DeepMind, [2]National Unviersity of Singapore
karkus@comp.nus.edu.sg

ABSTRACT

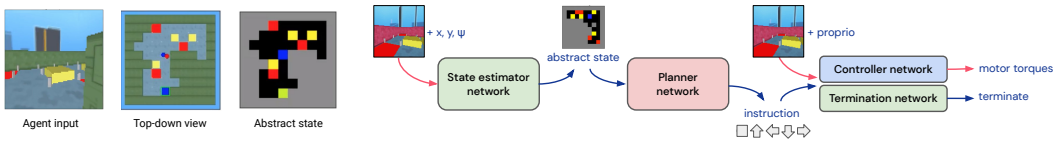
Embodied agents need to achieve abstract objectives using concrete, spatiotemporally complex sensory information and motor control. *Tabula rasa* deep reinforcement learning (RL) has tackled demanding tasks in terms of either visual, abstract, or physical reasoning, but solving these jointly remains a formidable challenge. To address this challenge, we propose a Modular RL approach that partitions embodied reasoning into specialized modules for state estimation, planning, and control. The modules are trained with independent RL objectives and training regimes, but they can adapt to each other during training. We show that Modular RL dramatically outperforms standard deep RL methods on Mujoban, a new, demanding domain that embeds Sokoban puzzles in physical 3D environments. Our results give strong evidence for the importance of research into modular designs: compared to black-box architectures, modular RL can more directly incorporate additional learning signals, choose more efficient training regimes, and can more flexibly adapt to changes in the task.

1 INTRODUCTION

Embodied domains, e.g., a robot performing a household task such as cooking or tidying a room, are long-standing challenges for AI research, simultaneously requiring complex perception due to partial observability, abstract long-horizon planning, and continuous high-dimensional control. Two paradigms have dominated the design of agents in this space. Classical robotics decomposes the overall controller into separately designed subsystems (Arkin, 1998). This allows injecting domain knowledge and offers reusable components; however, the overall systems can be brittle because of imperfect models, interfaces, or approximate inference. In contrast, deep RL optimizes agent performance end-to-end, directly for the task, but it typically takes a *tabula rasa* approach aiming for structures as general as possible. This has led to breakthroughs in domains with limited expert intuition; however, without structure, deep RL can perform worse in terms of both data efficiency, final performance and generalization than methods with well-designed, domain-specific structure (Kansky et al., 2017; Zambaldi et al., 2019). This effect is especially pronounced on embodied domains, where the need to *jointly* learn perception, multi-step reasoning, and long-horizon continuous control often renders standard exploration and learning strategies ineffective.

In this paper we propose a deep modular RL approach that partitions embodied reasoning into distinct RL modules. The approach combines structural ideas from robotics with architecture designs and optimization techniques from RL. We revisit one of the oldest structural ideas in robotics, the sense-plan-act (SPA) modular architecture (see Arkin 1998, Ch. 4), where separate modules perform state estimation ("sense"), abstract reasoning ("plan"), and low-level motor control ("act"). We update the SPA architecture with modular components that can be trained with RL (Fig. 1(b)). The modular RL approach similarly decomposes an overall RL system into submodules with well-defined roles and independent training regimes, thus taming the complexity of the overall task, but at the same time it leverages powerful deep neural networks trained with task-oriented RL.

We investigate our approach on Mujoban, a new domain inspired by the classical Sokoban puzzle, where an agent is tasked with organizing a warehouse by moving boxes around (Fig. 1(a)). Mujoban is designed to capture the general embodied reasoning problem as faced in robotics, combining partial



(a) Mujoban domain (b) Modular RL architecture

Figure 1: **(a)** Mujoban domain that embeds Sokoban puzzles in physical 3D environments. Left: the agent receives a first-person partial view. Middle: top-down view (not observed). Right: abstract state that corresponds to the underlying Sokoban state (not observed). **(b)** Modular RL architecture that partitions the policy into separate RL modules for state-estimation, planning and control.

observability (due to the agent sensing the world in the first person), irreversible actions and abstract long term reasoning (1000-4800 steps long episodes with 50-100 Sokoban moves), and low-level locomotion and object manipulation. A standard monolithic RL method fails to solve all but the easiest versions of this task, while our modular RL system solves most of the hard levels. We also show preliminary results on module transfer: learned abstract reasoning transfers when the 2-DOF ball body is replaced with a 8-DOF ant body. Our results reaffirm the benefit of prior knowledge in form of modular structural biases and suggest the importance of research in this direction.

2 Deep Modular Reinforcement Learning

We propose a modular RL approach that separates layers of embodied reasoning into distinct RL modules inspired by common robotics pipelines: a state estimation network for spatial reasoning with partial observations; a planner network for long-horizon abstract reasoning; a control network for goal-oriented physical reasoning. In our approach modules are trained in different regimes for different objectives, and are composed into a single policy using fixed interfaces. We train modules in a sequence, treating other modules as part of the environment, but at the same time allowing modules to adapt to each other in the context of the overall task. We explore the approach in the Mujoban domain. The domain and the modular architecture are shown in Fig. 1(a) and Fig. 1(b), respectively.

Mujoban domain. Mujoban is a new simulation domain that embeds Sokoban puzzles ((Junghanns & Schaeffer, 1997; Racanière et al., 2017; Guez et al., 2019)) in the Mujoco simulator (Todorov et al., 2012). Mujoban generates 3D maze equivalents of Sokoban levels with varied visual appearance, where an embodied agent navigates the maze and pushes boxes onto target locations using a physical 2-DOF ball body. The agent receives partial observations in the form of first-person camera images o^{cam}, standard proprioceptive observations o^{pr}, and the agent's absolute pose $o^{pos} = (x, y, \psi)$. The simulator also provides access to additional observations, e.g. for training: top-down camera images o^{top}, absolute pose of boxes o^{box}, and abstract state s^{abst} that represents the underlying Sokoban state as a $N \times N \times 4$ binary image. Details are in Section B of the Appendix.

State estimator. The state estimator module builds an abstract representation of the environment, $\hat{s}_t^{\text{abst}}$, from first-person visual observations o_t^{cam} and agent poses o_t^{pos}. The task is similarly to *mapping*. The map here is a $10 \times 10 \times 4$ abstract representation trained to predict the underlying Sokoban state, s_t^{abst} (see Fig. 1(a) for an example). We collect data by executing the overall policy. We train the module using cross-entropy losses and the true abstract states s_t^{abst} as labels. The network is a ConvLSTM with a geometry-aware frame processor that works intuitively by attending to different elements of the visual input for each possible location of the abstract state.

Planner. The planner is a time-abstracted RL policy for solving the underlying puzzle. It maps from abstract states $\hat{s}_t^{\text{abst}}$ to abstract instructions A_t. We design instructions using some understanding of the domain, namely, that moving and pushing boxes between cells is sufficient to succeed. Instructions are move north, east, south, west, or stay. More generally instructions could encode target goals in the abstract state. The policy network is a feed-forward adaptation of the repeated ConvLSTM architecture of Guez et al. (2019). We train with the MPO algorithm (Abdolmaleki et al., 2018) adapted to the time-abstracted discrete RL task. In each step of the time-abstracted task the controller executes an instruction through multiple real environment steps. Rewards are defined as the sum of real environment rewards collected during the execution of the instruction. Failure to complete an instruction is treated as an early-termination of the time-abstracted episode.

RL Approach	Setting	Reward	Succ.
Modular (ours)		**11.42** (0.22)	**78.7%**
Monolithic, asym.		0.31 (0.02)	0.0%
Monolithic, sym.		1.62 (0.08)	1.5%
Monolithic, asym.	curriculum	0.32 (0.03)	0.0%
Monolithic, sym.	curriculum	0.79 (0.04)	0.2%
Fully obs., monolithic	curriculum | input o^{top}	5.05 (0.25)	27.9%
Fully obs., structured	curriculum | input $o^{\text{top}}, s^{\text{abst}}$	11.22 (0.23)	77.0%

Table 1: Main results

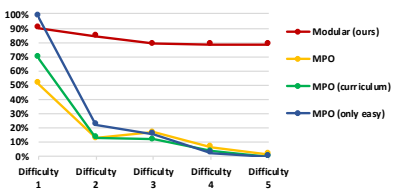


Figure 2: Success rates vs. level difficulty.

Controller. The controller module is a goal-oriented RL policy for visuomotor control, involving locomotion and object manipulation. The policy takes in an abstract instruction A_t, a visual observation o_t^{cam}, proprioceptive observations o_t^{pr}, and a termination signal β_t; and outputs continuous motor torques a_t^{τ}. We use a standard ResNet-LSTM network. We train with an asymmetric variant of MPO, where the critic network receives privileged information in the form of a goal-oriented representation of the agent pose o_t^{pos} and box poses o_t^{box}. In the controller RL task the agent gets a positive reward when an instruction is completed, and negative reward is given the instruction is not completed within $T_{instr} = 120$ steps, as well as a small negative reward each step the agent is moving backwards.

Control-termination. The control-termination module is a binary classifier trained to set β_t when the controller completes an instruction. The role of β_t is similar to option termiantion in hierarchical RL (Sutton et al., 1999; Barreto et al., 2019). Here an instruction is completed when the physical state corresponds to the instructed abstract state with some tolerance. We utilize β_t for time abstraction: the planner only updates the instruction if the last instruction has been completed ($\beta_t = 1$). We use a ResNet-LSTM network and train with a supervised cross-entropy loss.

Training. Modules are trained for independent objectives but they rely on each other to collect meaningful experience. We train modules in a bottom-up order: first the controller, using random instruction inputs; then the planner, using ground-truth abstract state inputs and executing the pre-trained controller; finally we train the state-estimator and control-termination modules together, executing the planner and controller. Training modules in parallel would be also possible in the future, as well as fine-tuning the entire architecture towards the sole optimization of the task reward.

3 Experiments

Our experiments aim to answer the following questions: 1) can we address a demanding embodied task by composing RL modules in our modular RL approach? 2) how do standard deep RL methods compare to Modular RL? 3) are alternative inductive biases, such as training curriculum, privileged information or structural policy priors, effective in this domain? 4) are trained modules reusable when some aspects of the task change?

Main results. Main results are summarized in Table 1 and Fig. 2. Additional results are in Section A. Illustrative videos can be found at `https://sites.google.com/view/modular-rl/`. We train models until near convergence, up to 7, 21 and 5 days for modules, fully-observable, and partially-observable baselines, respectively. An episode is successful if solved within 240s (4800 steps). Results show the best architectures and hyper-parameters, evaluated in 512 unseen random levels.

Modular vs. monolithic RL. The top rows of Table 1 compare modular and standard monolithic RL approaches in the most challenging, partially observable setting of Mujoban, with only egocentric observations. Success rates over difficulty levels are shown in Fig. 2. Monolithic approaches are trained with MPO, either using a shared torso for the policy and critic networks (sym.), or an asymmetric critic (asym.) that has the same set of privileged inputs as the Modular RL. They are trained only on hard levels like Modular RL (default); or alternatively on a mixture of level difficulties (curriculum), or only easy levels with one box (only easy). We observe that Modular RL achieves strong performance on the hardest levels. Monolithic MPO can learn easy levels, but they struggle with harder ones. The asymmetric MPO cannot take advantage of the same privileged observations as Modular RL; and curriculum only helps on easy levels but not on harder ones.

Fully observable baselines. The bottom rows of Table 1 reports agents in a simplified, fully observable setting of the task, that does not require mapping. The monolithic agent receives top-down camera observations o^{top}, and is trained with V-trace (Espeholt et al., 2018), using curriculum. Still, the agent can only solve less than 30% of the levels. This suggests that RL with structured policy priors for mapping but without time-abstracted planning, like Parisotto & Salakhutdinov (2017);

Gupta et al. (2017), would not work well for partially observable Mujoban. Finally, the last row shows an approach with both top-down camera and abstract state inputs, trained with V-trace and structured exploration, where extra pseudo rewards are given for reaching one of the random adjacent states in the abstract space. The structured agent performs significantly better, although still slightly below the modular approach with first-person only inputs.

Qualitative analysis. We observe that our modular agent quickly builds good abstract representations, moves around and pushes boxes, and solves many levels that are hard even for a human player. Failures can be attributed to imperfect abstract states (leading to poor irreversible moves); the planner failing to solve the level; and the controller failing an instruction when grid pegs block the motion of a (previously poorly aligned) box. Unlike the modular agent, the fully observable structured policy learned to exploit shortcuts, e.g., recover boxes pushed against a wall, or push multiple boxes. This suggests that in a stricter version of the task the benefit of our modular approach would be larger.

Additional results. Module-wise evaluation results are shown in Table 2 of the Appendix, where we replace modules with ground-truth oracle equivalents. We observe a larger performance gap for the learned planner module, and a smaller performance gap for the state-estimator module. Learning curves for each module are reported in Fig. 4. The rate of improvement of the learning curves indicate that reasonable performance can be achieved with less training data as well.

Module transfer. We experiment with the Mujoco ant in fully-observable Mujoban. Results are in Fig. 5 of the Appendix. We train a new controller module for the ant independently, and replace the existing controller without re-training the planner. Results show that despite the significantly harder control problem, success rates with the ant are only slightly lower than with the ball body (73.7% vs. 81.8%). This demonstrates an important benefit of the modular design: we can generalize and reuse parts of the solution if certain aspects of the task change, without having to relearn the rest.

4 Discussion

Deep RL has achieved dramatic success in components of embodied reasoning, e.g., abstract reasoning (Silver et al., 2016; Berner et al., 2019; Vinyals et al., 2019), perception, (Levine et al., 2016; Merel et al., 2019a; Gregor et al., 2019), and control (Heess et al., 2017; Gupta et al., 2019; Andrychowicz et al., 2020). However, few has attempted problems with all components of embodied intelligence. Notable exceptions are, e.g., Akkaya et al. (2019) and Lee et al. (2019). Many recent works combine structure with deep RL, in the form of loss (Jaderberg et al., 2017; Parisotto & Salakhutdinov, 2018; Gregor et al., 2019); algorithmic structure (Tamar et al., 2016; Farquhar et al., 2018; Racanière et al., 2017; Karkus et al., 2019); hierarchy (Dayan & Hinton, 1993; Gregor et al., 2016; Heess et al., 2016; Merel et al., 2019b); or modular decomposition (Andreas et al., 2016; Nachum et al., 2018; Zhang et al., 2018; Alet et al., 2018). A more extensive literature review is in Section E of the Appendix.

Our results show that coupling RL with an appropriately designed modular learning architecture can lead to progress on domains that are off-puttingly difficult for blackbox approaches. Given enough capacity, computation and experience, a monolithic end-to-end architecture could of course learn to solve Mujoban. We view a modular design as facilitating the search over neural architecture and learning framework to make progress in the face of an intractable RL task: the interpretability of the modules can afford more informed design interventions, hyper-parameter tuning can be factorized by modules, and additional learning signals can be incorporated in a more direct way.

At the same time, pre-conceived modular structure may require additional engineering, additional information at training time, it may overfit to a domain, and might even constrain the agent from achieving the optimal solution. This pitfall can be avoided by designing the modules to be as general as possible and by allowing joint refinement. For embodied tasks, we argue that inherent structural properties can be duly exploited: the world is spatially organized, contains persistent objects, and the agent perceives and acts locally from a single physical location. In this spirit, future work should continue to probe the boundary between module engineering and blackbox RL to characterize the best trade-offs between domain-specific engineering and a tabula rasa design.

The modular approach was successful in Mujoban, perhaps since layers of reasoning can be well separated in this domain. Going forward, we want to investigate the same modular philosophy in scenarios with blurrier decompositions, where joint fine-tuning with respect to the final task objective could follow independent training to compensate for potentially incorrect assumptions.

REFERENCES

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

Abbas Abdolmaleki, Jost Tobias Springenberg, Yuval Tassa, Remi Munos, Nicolas Heess, and Martin Riedmiller. Maximum a posteriori policy optimisation. In *ICLR*, 2018.

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

Ferran Alet, Tomás Lozano-Pérez, and Leslie P Kaelbling. Modular meta-learning. In *CoRL*, 2018.

Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. Differentiable MPC for end-to-end planning and control. In *Advances in Neural Information Processing Systems*, pp. 8299–8310, 2018.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *CVPR*, 2016.

Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

Ronald C. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *AAAI*, 2017.

André Barreto, Diana Borsa, Shaobo Hou, Gheorghe Comanici, Eser Aygün, Philippe Hamel, et al. The option keyboard: Combining skills in reinforcement learning. In *NeurIPS*, 2019.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Dkebiak, Christy Dennison, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Rodney A. Brooks. A robust layered control system for a mobile robot. *AI Memo 864, Artificial Intelligence Lab MIT*, 1985.

Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In *NeurIPS*, 1993.

Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *CoRL*, 2017.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, et al. IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures. In *ICML*, 2018.

Gregory Farquhar, Tim Rocktäschel, Maximilian Igl, and Shimon Whiteson. TreeQN and ATreeC: Differentiable tree planning for deep reinforcement learning. In *ICLR*, 2018.

Aleksandra Faust, Kenneth Oslund, Oscar Ramirez, Anthony Francis, Lydia Tapia, Marek Fiser, and James Davidson. PRM-RL: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *ICRA*, 2018.

N. Fazeli, M. Oller, J. Wu, Z. Wu, J. B. Tenenbaum, and A. Rodriguez. See, feel, act: Hierarchical learning for complex manipulation skills with multisensory fusion. *Science Robotics*, 4(26), 2019.

Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *ICRA*, 2017.

Chelsea Finn, Xin Yu Tan, Yan Duan, Trevor Darrell, Sergey Levine, and Pieter Abbeel. Deep spatial autoencoders for visuomotor learning. In *ICRA*, 2016.

Erann Gat. On three-layer architectures. In *Artificial Intelligence and Mobile Robots*, 1997.

Karol Gregor, Danilo Jimenez Rezende, and Daan Wierstra. Variational intrinsic control. 2016.

Karol Gregor, Danilo Jimenez Rezende, Frederic Besse, Yan Wu, Hamza Merzic, and Aaron van den Oord. Shaping belief states with generative environment models for RL. In *NeurIPS*, 2019.

Arthur Guez, Mehdi Mirza, Karol Gregor, Rishabh Kabra, Sébastien Racanière, Théophane Weber, et al. An investigation of model-free planning. In *ICML*, 2019.

Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. In *CoRL*, 2019.

Saurabh Gupta, James Davidson, Sergey Levine, Rahul Sukthankar, and Jitendra Malik. Cognitive mapping and planning for visual navigation. *arXiv preprint arXiv:1702.03920*, 2017.

David Ha and Jürgen Schmidhuber. Recurrent world models facilitate policy evolution. In *NeurIPS*, 2018.

Danijar Hafner, Timothy Lillicrap, Ian Fischer, Ruben Villegas, David Ha, Honglak Lee, and James Davidson. Learning latent dynamics for planning from pixels. In *ICML*, 2019.

Jessica B Hamrick, Andrew J Ballard, Razvan Pascanu, Oriol Vinyals, Nicolas Heess, and Peter W Battaglia. Metacontrol for adaptive imagination-based optimization. In *ICLR*, 2017.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

Nicolas Heess, Jonathan J. Hunt, Timothy P. Lillicrap, and David Silver. Memory-based control with recurrent neural networks. In *NeurIPS Deep Reinforcement Learning Workshop*, 2015.

Nicolas Heess, Gregory Wayne, Yuval Tassa, Timothy P. Lillicrap, Martin A. Riedmiller, and David Silver. Learning and transfer of modulated locomotor controllers. *arXiv preprint arXiv: 1610.05182*, 2016.

Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, et al. Emergence of locomotion behaviours in rich environments. *arXiv preprint arXiv:1707.02286*, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

Max Jaderberg, Volodymyr Mnih, Wojciech Czarnecki, Tom Schaul, Joel Z Leibo, David Silver, and Koray Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*, 2017.

Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, et al. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

Michael Janner, Sergey Levine, William T Freeman, Joshua B Tenenbaum, Chelsea Finn, and Jiajun Wu. Reasoning about physical interactions with object-oriented prediction and planning. In *ICLR*, 2019.

Andreas Junghanns and Jonathan Schaeffer. Sokoban: A challenging single-agent search problem. In *IJCAI Workshop on Using Games as an Experimental Testbed for AI Reasearch*, 1997.

Lukasz Kaiser, Mohammad Babaeizadeh, Piotr Milos, Blazej Osinski, Roy H Campbell, Konrad Czechowski, et al. Model-based reinforcement learning for Atari. In *ICLR*, 2020.

Ken Kansky, Tom Silver, David A. Mély, Mohamed Eldawy, Miguel Lzaro-Gredilla, Xinghua Lou, et al. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. In *ICML*, 2017.

Peter Karkus, Xiao Ma, David Hsu, Leslie Pack Kaelbling, Wee Sun Lee, and Tomás Lozano-Pérez. Differentiable algorithm networks for composable robot learning. In *RSS*, 2019.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Tejas D Kulkarni, Ankush Gupta, Catalin Ionescu, Sebastian Borgeaud, Malcolm Reynolds, Andrew Zisserman, and Volodymyr Mnih. Unsupervised learning of object keypoints for perception and control. In *NeurIPS*, 2019.

Youngwoon Lee, Edward S Hu, Zhengyu Yang, Alex Yin, and Joseph J Lim. IKEA furniture assembly environment for long-horizon complex manipulation tasks. *arXiv preprint arXiv:1911.07246*, 2019.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *JMLR*, 17(1):1334–1373, 2016.

Jeongsoo Lim, Inho Lee, Inwook Shim, Hyobin Jung, Hyun Min Joe, Hyoin Bae, et al. Robot system of DRC-HUBO+ and control strategy of team KAIST in DARPA robotics challenge finals. *Journal of Field Robotics*, 34(4), 2017.

Siqi Liu, Guy Lever, Josh Merel, Saran Tunyasuvunakool, Nicolas Heess, and Thore Graepel. Emergent coordination through competition. *arXiv preprint arXiv:1902.07151*, 2019.

Josh Merel, Arun Ahuja, Vu Pham, Saran Tunyasuvunakool, Siqi Liu, Dhruva Tirumala, Nicolas Heess, and Greg Wayne. Hierarchical visuomotor control of humanoids. In *ICLR*, 2019a.

Josh Merel, Saran Tunyasuvunakool, Arun Ahuja, Yuval Tassa, Leonard Hasenclever, Vu Pham, et al. Reusable neural skill embeddings for vision-guided whole body movement and object manipulation. *arXiv preprint arXiv: 1911.06636*, 2019b.

K. Mohta, K. Sun, S. Liu, M. Watterson, B. Pfrommer, J. Svacha, Y. Mulgaonkar, C. J. Taylor, and V. Kumar. Experiments in fast, autonomous, GPS-denied quadrotor flight. In *ICRA*, 2018.

Ofir Nachum, Shixiang (Shane) Gu, Honglak Lee, and Sergey Levine. Data-efficient hierarchical reinforcement learning. In *NeurIPS*, 2018.

N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 1980.

Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. In *NeurIPS*, 2017.

Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. *arXiv preprint arXiv:1702.08360*, 2017.

Emilio Parisotto and Ruslan Salakhutdinov. Neural map: Structured memory for deep reinforcement learning. In *ICLR*, 2018.

Sébastien Racanière, Théophane Weber, David Reichert, Lars Buesing, Arthur Guez, Danilo Jimenez Rezende, Adria Puigdomenech Badia, Oriol Vinyals, Nicolas Heess, Yujia Li, et al. Imagination-augmented agents for deep reinforcement learning. In *NeurIPS*, 2017.

Manolis Savva, Abhishek Kadian, Oleksandr Maksymets, Yili Zhao, Erik Wijmans, Bhavana Jain, et al. Habitat: A platform for embodied AI research. In *ICCV*, 2019.

Julian Schrittwieser, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, et al. Mastering Atari, Go, Chess and Shogi by planning with a learned model. *arXiv preprint arXiv:1911.08265*, 2019.

Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *NeurIPS*, 2015.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017a.

David Silver, Hado van Hasselt, Matteo Hessel, Tom Schaul, Arthur Guez, Tim Harley, et al. The predictron: End-to-end learning and planning. In *ICML*, 2017b.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, et al. A general reinforcement learning algorithm that masters Chess, Shogi, and Go through self-play. *Science*, 362 (6419):1140–1144, 2018.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181 – 211, 1999.

Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. In *NeurIPS*, 2016.

Sebastian Thrun, Mike Montemerlo, Hendrik Dahlkamp, David Stavens, Andrei Aron, James Diebel, Philip Fong, et al. Stanley: The robot that won the DARPA Grand Challenge. *Journal of Field Robotics*, 23(9): 661–692, 2006.

Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IROS*, 2012.

Rishi Veerapaneni, John D Co-Reyes, Michael Chang, Michael Janner, Chelsea Finn, Jiajun Wu, Joshua B Tenenbaum, and Sergey Levine. Entity abstraction in visual model-based reinforcement learning. In *CoRL*, 2019.

Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. FeUdal networks for hierarchical reinforcement learning. In *ICML*, 2017.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7587):350–354, 2019.

Nicholas Watters, Loic Matthey, Matko Bosnjak, Christopher P. Burgess, and Alexander Lerchner. COBRA: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration. *arXiv preprint arXiv:1905.09275*, 2019.

Greg Wayne, Chia-Chun Hung, David Amos, Mehdi Mirza, Arun Ahuja, Agnieszka Grabska-Barwinska, et al. Unsupervised predictive memory in a goal-directed agent. *arXiv preprint arXiv:1803.10760*, 2018.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, et al. Deep reinforcement learning with relational inductive biases. In *ICLR*, 2019.

Amy Zhang, Adam Lerer, Sainbayar Sukhbaatar, Rob Fergus, and Arthur Szlam. Composable planning with attributes. *arXiv preprint arXiv:1803.00512*, 2018.

A ADDITIONAL RESULTS

A.1 EXAMPLE TRAJECTORY

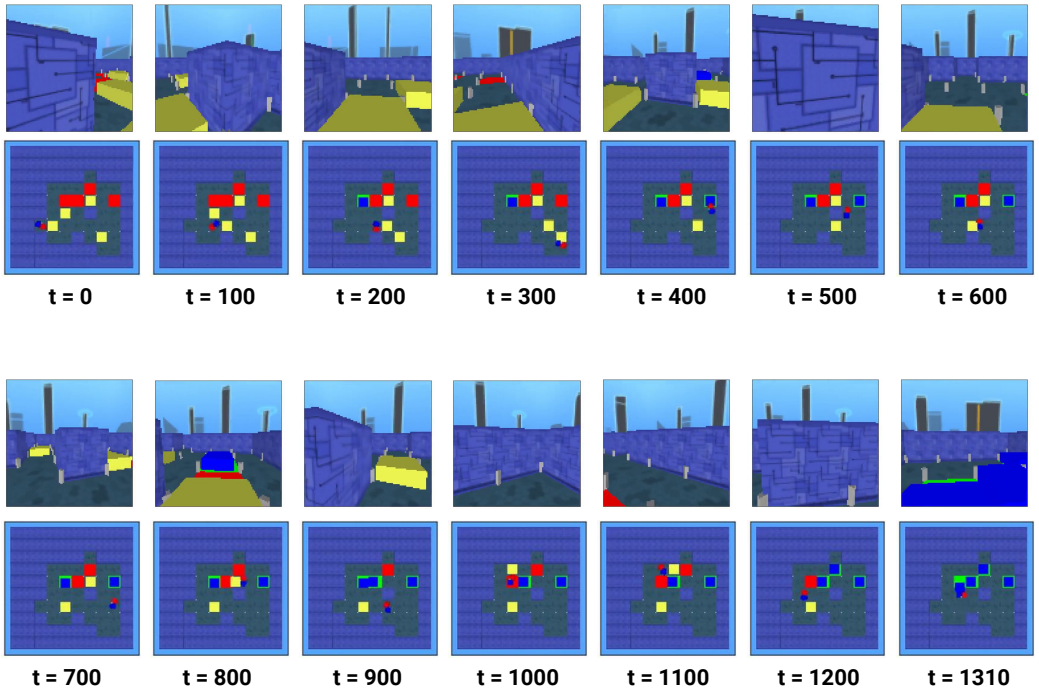


Figure 3: Modular RL example trajectory.

Fig. 3 shows a successful evaluation run of the Modular RL agent. The top rows show the first-person view input to the agent. The bottom row shows a top-down view of the environment, that is not observed by the agent. From the top-down view the agent appears as two colored discs, blue and red on the left and right side of the agent, respectively. More examples are available at `https://sites.google.com/view/modular-rl/`.

A.2 MODULE LEARNING RESULTS

Modular RL configuration	**Extra input**	**Reward**	**Succ.**
default		11.42 (0.22)	78.7%
planner + controller only	oracle s^{abst}, β	11.83 (0.20)	81.8%
controller only	oracle A, β	13.33 (0.12)	94.3%

Table 2: Module evaluation results.

Table 2 reports evaluation results for modules in different configurations. Rows of the table replaces modules with ground-truth oracle equivalents , which shows a larger performance gap for the learned planner module, and a smaller performance gap for the state-estimator module.

In Fig. 4 we report learning curves for each module, plotting the relevant objective metrics (rewards or accuracy) against environment steps sampled from the replay buffer.We select results for the best hyper-parameter and random seed (out of 2). Although we continued training each module for 7 days, learning curves indicate that reasonable performance can be achieved with less training as well.

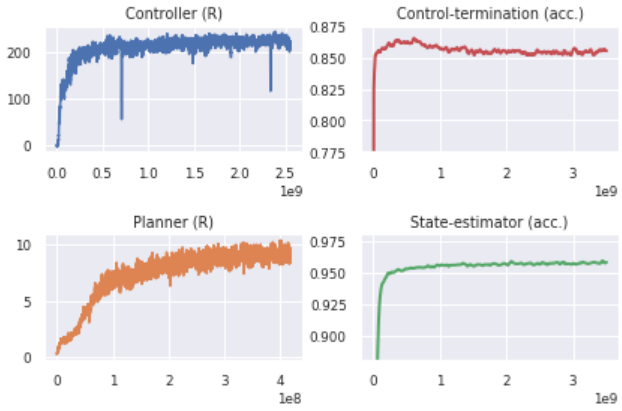


Figure 4: Modular RL learning curves (training objective vs. sampled environment steps).

A.3 Module transfer: ant body in Mujoban

Fig. 5 reports Modular RL results using ball and ant bodies in Mujoban. To focus on the significantly increased control complexity, in these experiments we only use the planner and controller modules and assume that abstract states are observed. We train a controller module for the ant independently, and swap out the controller module without training the planner. Results show that despite the significantly more challenging motor control problem, success rates with the ant are only slightly lower than with the ball body. This demonstrates an important benefit of the modular design and fixed interfaces: we can generalize and reuse parts of the solutions when only certain aspects of the task change, without having to relearn the rest.

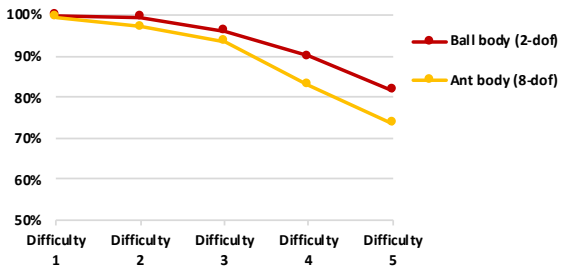


Figure 5: Modular RL results for different physical bodies in fully-observable Mujoban (success rates vs. difficulty level).

For the Ant body the physical environment is scaled $3\times$, however the weight of boxes and the size of grid pegs remain unchanged. To account for the larger environment and slower motion of the Ant, we also increased the time limits from 240s to 480s. The controller for the Ant has been trained for 3 weeks. For evaluation we simply replace the controller trained for the ball body to the controller trained for the ant body, without retraining the planner module.

B Mujoban domain

Overview

Mujoban is a simulation domain that embeds Sokoban puzzles in Mujoco simulation environment Todorov et al. (2012). Sokoban is a popular RL benchmark with complex planning Junghanns & Schaeffer (1997); Racanière et al. (2017); Guez et al. (2019), where the agent pushes boxes onto target locations in procedurally generated 2D grids. Mujoban generates 3D maze equivalents of Sokoban levels with varied visual appearance. An embodied agent navigates the maze given partial visual observations, and manipulates boxes using its physical body.

The Mujoban domain is visualized in Fig. 1(a). In the default configuration the agent has a 2-DOF ball body; and it receives partial observations in the form of first-person camera images o^{cam}, standard proprioceptive observations o^{pr}, and the agent's absolute pose $o^{pos} = (x, y, \psi)$. The simulator also provides access to additional observations, e.g. for training: top-down camera images o^{top}, absolute pose of boxes o^{box}, and abstract state s^{abst} that represents the underlying Sokoban state as a $N \times N \times 4$ binary image, where image channel correspond to the presence of wall, target, agent, and box in each of the grid locations.

Mujoban exhibits key challenges of embodied reasoning: partial observability, long-horizon reasoning, continuous control and object manipulation. Compared to other, possibly more realistic embodied RL domains like Lee et al. (2019); Savva et al. (2019); Dosovitskiy et al. (2017), an important benefit of Mujoban is the ability to separately control the difficulty of each layer of embodied reasoning. Specifically, we can control the abstract reasoning complexity by changing the size of the underlying Sokoban level and the number of boxes; state estimation complexity by adding top-down or direct state observations; and control complexity by replacing the agent's body e.g. by replacing the ball body with the Mujoco ant.

The environment provides different levels of difficulty. In our experiments we aim for the hardest, partially observable configuration of the task: 10×10 Sokoban levels with 4 boxes, first-person observations $(o^{\mathrm{cam}}, o^{\mathrm{pr}}, o^{pos})$. We use easier levels only for evaluation, and leverage extra observations only during training (and for baselines).

MUJOBAN VS. SOKOBAN

A box is considered to be on target when at least 50% of both its axis are within a target pad boundary The color and texture of tiles on floor and walls are randomized in each episode. There are skylines in the background which are kept fixed and help the agent for global orientation.

The physical body of the agent is modified version of BoxHead walker[1] Liu et al. (2019). The arms and kick actuator are removed. In addition the walker is equipped with red and blue geom balls on its head in order for the agent to be able to infer orientation from images. The body is controlled at 20Hz. The simulator also allows for using alternative bodies defined in Mujoco, such as the ant.

To make the underlying logic of Mujoban similar to Sokoban, pegs are inserted at each grid point, acting as physical barriers that confine the motion of boxes to horizontal and vertical axes (and make manipulating boxes harder). Yet the correspondence with Sokoban rules is not perfect, e.g., there is no built-in mechanism that prevents pushing multiple boxes, and although challenging, getting a box off a wall is also conceivable.

REWARDS

Rewards are similar to Sokoban: +1 for pushing a box on a target pad, -1 for removing a box from a target pad, and +10 for solving the level.

LEVELS

Table 3 shows the levels configuration. We pad all rooms with walls so the final size of all levels are 10×10.

Level	Grid size	Number of boxes
1	5×5	1
2	7×7	1
3	7×7	2
4	8×8	3
5	10×10	4

Table 3: Mujoban level settings

[1] `https://github.com/deepmind/dm_control/blob/master/dm_control/locomotion/soccer/boxhead.py`

C MODULAR RL DETAILS

C.1 STATE ESTIMATOR

The state estimator is a recurrent classifier mapping from $(o^{\text{cam}}_{t'}, o^{pos}_{t'})_{t' \leq t}$ and to s^{abst}_t. The state estimator is composed of three components: the first one is a neural network with geometry-aware structural prior, shown in Fig. 6, which pre-processes each visual frame separately. The output of the network is fed into a ConvLSTM Shi et al. (2015), which forms an implicit belief state over the abstract state. The output of the LSTM is input into a classifier with 5 labels (None,Wall,Box,Target,Box-on-target), which predicts each spatial location of the abstract state separately (an autoregressive classifier could also be used, but we found it not necessary).

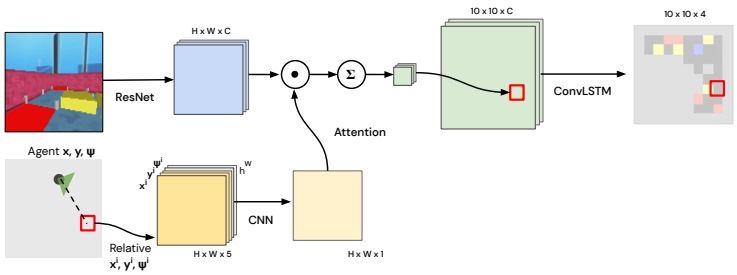


Figure 6: Geometry-aware state-estimator network.

The geometry-aware frame processor works intuitively by attending to different elements of the visual input for each possible location of the abstract state. More precisely, we first extract a $H \times W \times C$ feature tensor from o^{cam}_t using a ResNet He et al. (2016). Then, for each spatial location (x, y) of the abstract state s^{abst}_t, we use the agent pose input o^{pos}_t to compute the coordinates of the spatial location relative to the pose. The relative spatial coordinates are tiled and concatenated with pixel coordinates to a $H \times W \times C$ features tensor. We pass the resulting coordinate matrix through a CNN, which defines attention weights of size $H \times W \times 1$. We normalize weights to sum to one, and compute the weighted sum of image features over the H and W dimensions for each channel, which results in a single feature column $1 \times 1 \times C$. This is done for each element (x, y) of the abstract state in parallel (using shared weights); the different outputs are then recombined into a $10 \times 10 \times C$ feature map.

The geometry-aware state-estimator relies on knowing the agent's pose. We used a setting where the pose is directly observed for simplicity, but it could be also predicted from visual and proprioceptive observations.

We collect data for training by executing the planner, controller and control-termination modules, and using true abstract states as labels. We treat the output of the state estimator as independent binary classifiers for each cell, separately for the box, target pad and wall layers of the abstract state. For the agent state layer we treat the output as a single classifier over all cells. The loss is then given by the sum of independent cross-entropy losses for each classifier.

C.2 PLANNER

The planner is a model-free RL policy for a time-abstracted RL task. Inputs are abstract state estimates, $\hat{s}^{\text{abst}}_t$, outputs are discrete instructions, A_t.

The planner policy is trained with MPO Abdolmaleki et al. (2018), a recent actor-critic off-policy RL algorithm. We adapt MPO to a time-abstracted discrete RL task. In each step of the time-abstracted task the controller module executes the abstract instruction through multiple real environment steps, as many as needed to complete the instruction. Rewards are defined as the sum of real environment rewards collected during the execution of the instruction, plus a -0.01 reward for each planner step. In case the instruction is not completed within $T_{instr} = 120$ steps, the experience is dropped and treated as an early-termination of the time-abstracted episode. During training we input ground-truth abstract states to the planner, which is replaced by predicted s^{abst}_t during evaluation.

We use a policy network similar to the repeated ConvLSTM architecture of Guez et al. (2019). The network is built only of generic components, but it has strong structural bias for playing Sokoban. We use a feed-forward variant of the architecture, replacing the hidden states by a trainable variable that is not propagated through time. We add a separate critic head to the architecture with 5 output values, the Q-value estimates for each discrete instruction.

C.3 CONTROLLER

The controller is a goal-oriented model-free policy trained with MPO. Inputs are abstract instruction A_t, visual observation o_t^{cam}, proprioceptive observations o_t^{pr}, and termination signal β_t. Outputs are continuous motor torques a_t^τ.

The policy network is a ResNet for processing the visual input, connected to an LSTM Hochreiter & Schmidhuber (1997) and a fully-connected layer that outputs parameters of a multivariate Normal distribution. We use an asymmetric actor-critic setup, where the critic network receives privileged information during training. The inputs to the critic are the agent pose o_t^{pos}, the pose of boxes o_t^{box}, proprioceptive observations o_t^{pr} and a control action a^τ. The output is a Q-value estimate for a^τ.

We define an RL environment for training the controller. The agent receives (random) instructions during training. It receives a positive reward ($+4$) when an instruction is completed, and a negative reward (-5) if the instruction is not completed within $T_{instr} = 120$ steps. Further, to encourage forward motion a small negative reward is given every step the agent is moving backwards. Rewards are computed using privileged state observations from the simulator.

C.4 CONTROL-TERMINATION

The control-termination module is a sequential binary classifier trained with supervised data. The inputs are o_t^{cam}, o_t^{pr}, A_{t-1} and β_{t-1}. The output is $p_t(\beta)$, the estimated probability that the controller has completed the last abstract instruction at time t. The network architecture is a convolutional ResNet connected to an LSTM. In our implementation ResNet weights are shared with the state estimator network. The termination-signal β_t is sampled according to $\beta_t \sim p_t(\beta)$. During evaluation a positive termination signal is added if there has been no termination predicted for 120 steps.

We train the control-termination module together with the state estimator by executing the planner and controller modules. During execution the predicted β signal is fed to the controller. Supervised training labels are obtained according to a strict definition of completing an abstract instruction (defined below), and computed using privileged observations available during training. We use a re-weighted cross-entropy loss that accounts for the imbalanced number of positive and negative samples. Specifically, we re-weight the loss with weight $w_t = 1 + N_{missed}$, where N_{missed} is the number of steps the prediction has been negative while the label has been positive.

The definition of completing an instruction is as follows. An instruction is completed if the agent is near the center of the target grid cell corresponding to the instructed move, with a $d_{tol} = 0.1$ unit tolerance. If the target cell is occupied by a box the box needs to be pushed to the center of the adjacent cell with $d_{tol} = 0.1$ tolerance. If the target cell is occupied by a wall the instruction is infeasible, and the agent must remain in its current grid cell. When training the controller we compute the completion criteria and rewards based on privileged state observations. When evaluating the system the termination signal is given by the β prediction.

C.5 EXPERIMENTAL SETUP

We train modules of Modular RL on randomly generated Mujoban levels from the most difficult category (10×10 grid with 4 boxes). After modules are trained we evaluate the full system on a separate set of 512 random levels. We use privileged observations of the true environment state during training but not during evaluation. An evaluation episode is successful if the level is solved within 240s (4800 environment steps). We also run evaluations on easier levels of Mujoban, with smaller grid and less boxes, but we do not use the easier levels during training unless indicated.

We used a distributed RL setup with one learner and N_{actor} actors. We choose N_{actor} for each module independently to keep the ratio of actor and learner steps for off-policy learning similar. We use $N_{\text{actor}} = 200$ for the state-estimator and control-termination modules, $N_{\text{actor}} = 1000$ for the planner

module, and $N_{\text{actor}} = 256$ for the controller module. We train modules until near convergence, which took up to 7 days. Baseline networks were trained in a similar distributed setup, while we continued training for 21 and 5 days, using $N_{\text{actor}} = 1000$ and $N_{\text{actor}} = 256$, for fully-observable and partially-observably baselines, respectively. For partially observable baselines we found in initial experiments that terminating episodes early during training ($T_{env} = 45s$) produced better results, hence we train them in this setting. For evaluation we increase the time limit to $T_{env} = 240s$, same as for Modular RL.

All models are implemented in Tensorflow (Abadi et al., 2015) and trained with the Adam optimizer (Kingma & Ba, 2014). We use learner nodes equipped with an Nvidia Tesla V100 GPU. Actor have access only to CPU cores.

D BASELINES

As an alternative to our modular design, we run MPO with standard monolithic network architectures and train end-to-end for the overall rewards. To gain better understanding of the task difficulty, we also consider a simplified setting and train agents with monolithic and structured architectures using the V-Trace Espeholt et al. (2018) algorithm. While the choice of RL algorithm, input observations, and network architecture are orthogonal, we did not try their combinations due to the substantial computation and time required to run these experiments (on the order of weeks).

D.1 BASELINES FOR PARTIALLY-OBSERVABLE MUJOBAN

Monolithic RL. We run MPO with standard network architectures that connect a ResNet to an LSTM. We search over network variants of three distinct sizes, as well as learning rates, discount factor, MPO parameters, and report the best setting. A small network variant is similar to the controller module except it does not receive instruction inputs. The medium network increases the LSTM hidden state size and the size of fully connected layers. The large network adds extra pre-processing layers for the non-visual inputs and extra layers after the LSTM output. We search over network variants and report best results.

Asymmetric vs. symmetric actor-critic. For a fair comparison, we first try an asymmetric actor-critic setting, where the critic network receives the same amount of privileged information we used for training Modular RL, i.e., the true abstract state s_t^{abst}, and poses o_t^{pos} and o_t^{box}. The policy network receives the same set of inputs as the modular policy: visual observation, agent pose, and proprioceptive observations. The critic network receives the same privileged information as we used for training Modular RL, i.e., the true abstract state s_t^{abst}, and physical state including the absolute pose of the agent and all boxes. We then also try a symmetric setup, where both actor and critic receive the same input. Here we use a single network torso with separate policy and Q-function network heads.

Curriculum. We train baselines only on hard levels, same as for Modular RL, as well as using a training curriculum with levels randomly sampled from all difficulty categories ($p_1 = 0.25, p_2 = 0.25, p_3 = 0.2, p_4 = 0.2, p_5 = 0.1$), and training only on easy levels ($p_1 = 1.0$). After training we evaluate for all difficulty categories.

D.2 BASELINES FOR FULLY-OBSERVABLE MUJOBAN

Fully-observable monolithic RL. Here we provide the agent with access to top-down camera images of the full environment, removing (most of) the partial observability. We first use a monolithic LSTM architecture and the V-trace Espeholt et al. (2018) algorithm. We train end-to-end for the overall RL task. We use curriculum with combination of all level difficulties.

The network details are as follows. Proprioception inputs are all concatenated and passed to one layer MLP with 100 hidden units. The vision inputs are passed through 3 layer ResNet with channels sizes 16, 32, 32 and each layer consists of 2 blocks. The outputs flattened and passed through one layer MLP with size of 256. All the flat inputs finally concatenated together and passed to and LSTM agent with similar architecture as the controller of our modular RL approach.

Fully-observable structured RL. Finally, we also try an alternative approach with strong structured exploration priors for Mujoban. Here the agent observes both top-down images and the true abstract state. We add structured exploration, i.e., add an extra pseudo reward of 0.1 for reaching one of the randomly chosen 4 adjacent state in the abstract space in a given time frame. This subgoal is given to the agent by inputting the target abstract state along with the current abstract state. The agent has extra value head and computes a V-trace loss separately for the pseudo reward and the sub-task episode. The gradients from this auxiliary loss are added to those associated with the main task, using a weight of 0.5 for the auxiliary loss. The aim of the auxiliary loss is to help the agent to explore more meaningfully in the abstract space. If the subgoal is not reached within 50 steps a new random subgoal is sampled. The network architecture is similar to the fully-observable monolithic agent. We train using the same curriculum as for monolithic baselines.

E BACKGROUND AND RELATED WORK

DEEP RL AND END-TO-END REASONING

Recent research in deep RL has led to dramatic progress in the components of embodied reasoning, including multi-step abstract reasoning Silver et al. (2016); Berner et al. (2019); Vinyals et al. (2019), egocentric perception and state estimation, e.g., Levine et al. (2016); Merel et al. (2019a;b); Gregor et al. (2019), and embodied spatiotemporal control Heess et al. (2017); Gupta et al. (2019); Andrychowicz et al. (2020) . In spite of this, and though few in the community deny the importance of the full embodied reasoning problem, little recent work has attempted to tackle problems with all components of embodied intelligence. Notable exceptions include solving the Rubik's cube using a robotic hand Akkaya et al. (2019), and the IKEA RL domain Lee et al. (2019). To contextualize our aims and approach, we review recent work in RL as it relates to solving full embodied reasoning tasks and contrast it to modular systems design in the wider control literature.

MODULAR SYSTEMS IN AI AND ROBOTICS

Although recent research in RL has largely focused on end-to-end solutions to behavioral problems, modular designs have a long history in artificial intelligence research. As early as the 1960s, researchers had designed systems with decomposable structure in order to solve challenging reasoning problems: see (Nilsson 1980, §1.4) for a discussion. The later robotics literature is full of modular systems that encompass perception and motor control in addition to abstract problem solving, e.g. Brooks (1985); Gat (1997). Modular design is one of the cornerstones of practical robotics systems because of the current intractability of end-to-end methods, the need for careful diagnosis of each component, and the desire for component reuse. The systems proposed for DARPA robotics challenge illustrate the general success of modular design patterns: current autonomous and semi-autonomous systems for controlling cars (Thrun et al., 2006), humanoid robots (Lim et al., 2017), and aerial vehicles (Mohta et al., 2018) consist of specialized modules for perception, reasoning, and actuator control coupled together with carefully designed interfaces.

MODULAR STRUCTURE IN DEEP RL

Although methods in deep RL often attempt to avoid engineering structure in favor of developing general methods, research into system structure has been a key component of deep RL's success. For example, consider the line of work leading from AlphaGo to MuZero (Silver et al., 2016; 2017a; 2018; Schrittwieser et al., 2019): the earliest models in this family included domain-specific engineering, such as data augmentation by board rotation and perfect knowledge of game rules. With the lessons learned by the success of earlier methods, later architectures progressively removed and modified structure - removing assumptions about game rules, refining the details of Monte Carlo tree search, and upgrading the convolutional architecture to residual networks - producing more general and more powerful methods. These later innovations were made possible by incorporating lessons learned by first engineering a more restrictively structured method for a very difficult problem of interest.

Compared to a monolithic system designed purely as a black box, a system that exploits modularity can exploit the system designer's intuition about how to structure a problem using specially designed architectures and losses. In this sense, many recently proposed methods have exploited modularity by engineering specialized architecture or losses: for example, by introducing structure exploiting

memory (Heess et al., 2015), map-based reasoning (Parisotto & Salakhutdinov, 2018), prediction (Jaderberg et al., 2017; Gregor et al., 2019), or several such components (Wayne et al., 2018; Jaderberg et al., 2019). This additional structure is typically chosen by appealing to the information content of the environment or correlations with the behavioral task of interest and empirically verified on a difficult benchmark.

Other related work aims to use algorithmic structures as priors for DNN policies (Tamar et al., 2016; Farquhar et al., 2018; Racanière et al., 2017; Amos et al., 2018). One recent work that combines the benefits of internal system structure with end-to-end learning is the Differentiable Algorithm Network (DAN) of Karkus et al. (2019). This model composes differentiable structures for state estimation (vision + filtering), planning, and control and trains them jointly for partially observed map-based navigation tasks. The resulting model can make plans that take into account global map structure while avoiding visually identified obstacles.

Other methods propose to learn a model which is then used to simplify the task of another model in some way. For instance, in hierarchical reinforcement learning (HRL), this may correspond to learning motor primitives (Heess et al., 2016; Merel et al., 2019b), learning hierarchies of agents which work at different resolution in time and space (Dayan & Hinton, 1993; Sutton et al., 1999; Vezhnevets et al., 2017; Bacon et al., 2017), or discovering structured options for exploration (Gregor et al., 2016).

In model-based RL, this corresponds to learning a model of the environment, which can then be leveraged by a controller in various ways, such as providing synthetic training data (Ha & Schmidhuber, 2018; Kaiser et al., 2020) or simulating trajectories to improve decision making at test time (Hamrick et al., 2017; Hafner et al., 2019; Faust et al., 2018), improving learning of policies during training (Silver et al., 2017b; Oh et al., 2017; Finn & Levine, 2017), or combining several of these aspects (Schrittwieser et al., 2019).

Yet another family of approach is to learn to predict structured representation of data and used for downstream reasoning and control; this includes methods trained to detect or segment visual structure and use the estimated structure for downstream reasoning (Finn et al., 2016; Kulkarni et al., 2019; Watters et al., 2019) and methods that leverage knowledge of object and physics simulators (Fazeli et al., 2019; Janner et al., 2019; Veerapaneni et al., 2019).

Finally, a line of research closely related to our proposed approach has sought to partition or structure reasoning, for instance by using an architecture combining reasoning in structured goal spaces with low-level policies (Nachum et al., 2018; Zhang et al., 2018; Alet et al., 2018) or by introducing sub-modules corresponding to a natural task partition (Andreas et al., 2016).