DEEP SETS FOR GENERALIZATION IN RL

Tristan Karch[1 2] **Cédric Colas**[1 2] **Laetitia Teodorescu** [2] **Clément Moulin-Frier** [2]

Pierre-Yves Oudeyer [2]

ABSTRACT

This paper investigates the idea of encoding object-centered representations in the design of the reward function and policy architectures of a language-guided reinforcement learning agent. This is done using a combination of object-wise permutation invariant networks inspired from Deep Sets and gated-attention mechanisms. In a 2D procedurally-generated world where agents targeting goals in natural language navigate and interact with objects, we show that these architectures demonstrate strong generalization capacities to out-of-distribution goals. We study the generalization to varying numbers of objects at test time and further extend the object-centered architectures to goals involving relational reasoning.

1 INTRODUCTION

Reinforcement Learning (RL) has begun moving away from simple control tasks to more complex multimodal environments, involving compositional dynamics and language. To successfully navigate and represent these environments, agents can leverage factorized representations of the world as a collection of constitutive elements. Assuming objects share properties (Green & Quilty-Dunn, 2017), agents may transfer knowledge or skills about one object to others. Just like convolutional networks are tailored for images, relational inductive biases (Battaglia et al., 2018) can be used to improve reasoning about relations between objects (e.g. in the CLEVR task, Johnson et al. (2017)). One example could be to restrict operations to inputs related to object pairs.

A companion paper described a setting where an agent that sets its own goals has to learn to interact with a set of objects while receiving descriptive feedbacks in natural language (NL) (Colas et al., 2020). This work introduced reward function and policy architectures inspired by Deep Sets (Zaheer et al., 2017) which operate on unordered sets of object-specific features, as opposed to the traditional concatenation of the features of all objects. In this paper, we aim to detail that contribution by studying the benefits brought by such architectures. We also propose to extend them to consider pairs of objects, which provides inductive biases for language-conditioned relational reasoning.

In these architectures, the final decision (e.g. reward, action) integrates sub-decisions taken at the object-level. Every object-level decision takes into account relationships between the body –a special kind of object– and either one or a pair of external objects. This addresses a core issue of language understanding (Kaschak & Glenberg, 2000; Bergen, 2015), by grounding the meaning of sentences in terms of affordant relations between one's body and external objects.

In related work, Santoro et al. (2017) introduced Relational Networks, language-conditioned relational architectures used to solve the supervised CLEVR task. Zambaldi et al. (2018) introduced relational RL by using a Transformer layer (Vaswani et al., 2017) to operate on object pairs, but did not use language. Our architectures also draw inspiration from gated attention mechanisms (Chaplot et al., 2017). Although other works also propose to train a reward function in parallel of the policy, they do so using domain knowledge (expert dataset in Bahdanau et al. (2019), environment dynamics in Fu et al. (2019)) and do not leverage object-centered representations.

Contributions - In this paper, we study the comparative advantage of using architectures based on factorized object representations for learning policies and reward functions in a language-conditioned RL setting. We 1) prove that our proposed architectures perform best in this setting

[1] for equal contribution

[2] Flowers Team, INRIA, France. Correspondence to: Tristan Karch <tristan.karch@inria.fr>

compared to non-factorized baselines, 2) study their capacity to allow generalization to out-of-distribution goals and generalization to additional objects in the scene at test time, and 3) show that this architecture can be extended to deal with goals related to object pairs.

2 PROBLEM DEFINITION

A learning agent explores a procedurally generated 2D world containing objects of various types and colors. Evolving in an environment where objects share common properties, the agent can transfer knowledge and skills between objects, which enables *systematic generalization* (e.g. *grasp green tree* + *grasp red cat* $\rightarrow$ *grasp red tree*). The agent can navigate in the 2D plane, grasp objects and grow some of them (animals and plants). A simulated social partner (SP) provides NL labels when the agent performs interactions that SP considers interesting (e.g. *grasp green cactus*). Descriptions are turned into targetable goals by the agent and used to train an internal reward function. Achievable goals $\mathcal{G}^{\text{A}}$ are generated according to the following grammar:

1. Go: *(e.g. go bottom left)*
   - *go* + ***zone***
2. Grasp: *(e.g. grasp red cat)*
   - *grasp* + ***color*** $\cup$ {*any*} + ***object type*** $\cup$ ***object category***
   - *grasp* + *any* + ***color*** + *thing*
3. Grow: *(e.g. grow green animal)*
   - *grow* + ***color*** $\cup$ {*any*} + ***living thing*** $\cup$ {*living_thing, animal, plant*}
   - *grow* + *any* + ***color*** + *thing*

Bold and { } represent sets of words while *italics* represents specific words, see detailed grammar in Section A.2. In total, there are 256 achievable goals, corresponding to an infinite number of scenes. These are split into a training set of goals $\mathcal{G}^{\text{train}}$ from which SP can provide feedbacks, and a testing set of goals $\mathcal{G}^{\text{test}}$ held out to test the generalization abilities of the agent. Although testing sentences are generated following the same composition rules as training sentences, they extend beyond the training distribution (*out-of-distribution generalization*). The agent can show two types of generalizations: from the reward function (*it knows when the goal is achieved*) and from the policy (*it knows how to achieve it*). Full details about the setup, architectures and training schedules are reported from Colas et al. (2020) in the Appendices.

Evaluation - Regularly, the agent is tested offline on goals from $\mathcal{G}^{\text{train}}$ (training performance) and goals from $\mathcal{G}^{\text{test}}$ (testing performance). We test both the average success rate ($\overline{SR}$) of the policy and the average F_1-score of the reward function over each set of goals. A goal's SR is computed over 30 evaluations. The F_1-score is computed from a held out set of trajectories (see Section C). Note that *training performance* refers to the performance on $\mathcal{G}^{\text{train}}$, but still measures *state generalization* as scenes are generated procedurally. In all experiments, we provide the mean and standard deviation over 10 seeds, and report statistical significance using a two-tail Welch's t-test at level $\alpha = 0.05$ as advised in Colas et al. (2019b).

3 DEEP SETS FOR RL

The agent learns in parallel a language model, an internal goal-conditioned reward function and a multi-goal policy. The language model ($\mathcal{LM}$) embeds NL goals ($\mathcal{LM}(g_{\text{NL}}) : \mathcal{G}^{\text{A}} \rightarrow \mathbb{R}^{100}$) using an LSTM (Hochreiter & Schmidhuber, 1997) trained jointly with the reward function via backpropagation (yellow in Fig. 1). The reward function, policy and critic become language-conditioned functions when coupled with $\mathcal{LM}$, which acts like a goal translator. The agent keeps tracks of goals discovered through its exploration and SP's feedbacks $\mathcal{G}_d$. It samples targets uniformly from $\mathcal{G}_d$.

Deep Sets - The reward function, policy and critic leverage modular architectures inspired by Deep Sets (Zaheer et al., 2017) combined with gated attention mechanisms (Chaplot et al., 2017). Deep Sets is a family of neural architectures implementing set functions (input permutation invariance). Each input is mapped separately to some (usually high-dimensional (Wagstaff et al., 2019)) latent space using a shared network. These latent representations are then passed through a permutation-invariant function (e.g. mean, sum) to ensure the permutation-invariance of the whole function.

Modular-attention architecture for the reward function - Learning a goal-conditioned reward function ($\mathcal{R}$) is framed as binary classification. The reward function maps a state $\mathbf{s}$ and a goal embedding $\mathbf{g} = \mathcal{LM}(g_{\text{NL}})$ to a binary reward: $\mathcal{R}(\mathbf{s}, \mathbf{g}) : \mathcal{S} \times \mathbb{R}^{100} \rightarrow \{0, 1\}$ (right in Fig. 1).

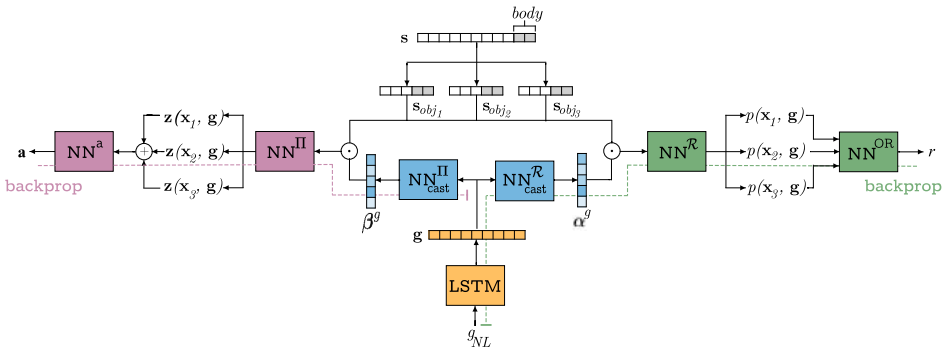


Figure 1: **Modular architectures with attention.** Left: policy. Right: reward function.

The reward function is constructed such that object-specific rewards are computed independently for each of the N objects before being integrated into a global reward through a logical OR function approximated by a differentiable network which ensures object-wise permutation invariance: *if any object verifies the goal, then the whole scene verifies it*. This *object-specific reward function* is shared for all objects ($\texttt{NN}^\mathcal{R}$). To evaluate a probability of positive reward p_i for object i, it needs to integrate both the corresponding object representation $\mathbf{s}_{obj(i)}$ and the goal. Instead of a simple concatenation, we use a gated-attention mechanism Chaplot et al. (2017). $\mathbf{g}$ is cast into an attention vector $\boldsymbol{\alpha}^g$ before being combined to $\mathbf{s}_{obj(i)}$ through an Hadamard product (term-by-term): $\mathbf{x}_i^g = \mathbf{s}_{obj(i)} \odot \boldsymbol{\alpha}^g$. The overall architecture is called *MA* for *modular-attention* and can be expressed by:

$$\mathcal{R}(\mathbf{s}, g) \;=\; \texttt{NN}^{\texttt{OR}}([\texttt{NN}^\mathcal{R}(\mathbf{s}_{obj(i)} \odot \boldsymbol{\alpha}^g)]_{i\in[1..N]}).$$

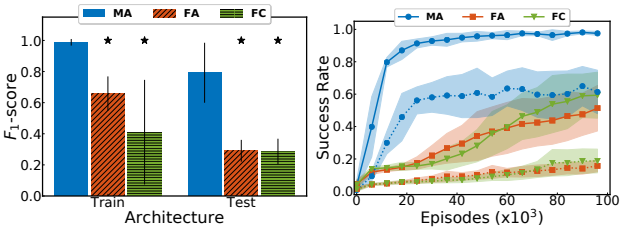
Modular-attention architecture for the policy and critic - Our agent is controlled by a goal-conditioned policy Π that leverages a *modular-attention (MA)* architecture (left in Fig. 1). Similarly, the goal embedding $\mathbf{g}$ is cast into an attention vector $\boldsymbol{\beta}^g$ and combined with $\mathbf{s}_{obj(i)}$ through a gated-attention mechanism. As usually done with Deep Sets, these inputs are projected into a high-dimensional latent space (of size $N \times dim(\mathbf{s}_{obj(i)})$) using a shared network $\texttt{NN}^\Pi$ before being summed. The result is finally mapped into an action vector $\mathbf{a}$ with $\texttt{NN}^\text{a}$. Following the same architecture, the critic (not shown in Fig. 1) computes the action-value of the current state-action pair given the current goal with $\texttt{NN}^\text{av}$:

$$\Pi(\mathbf{s}, g) \;=\; \texttt{NN}^\text{a}\Big(\sum_{i\in[1..N]} \texttt{NN}^\Pi(\mathbf{s}_{obj(i)} \odot \boldsymbol{\beta}^g)\Big), \qquad Q(\mathbf{s}, \mathbf{a}, g) \;=\; \texttt{NN}^\text{av}\Big(\sum_{i\in[1..N]} \texttt{NN}^Q([\mathbf{s}_{obj(i)}, \mathbf{a}] \odot \boldsymbol{\gamma}^g)\Big).$$

4 Experiments

4.1 Generalization Study

Figure 2 shows the training and testing performances of our proposed *MA* architectures and two baseline architectures: 1) *flat-concatenation (FC)* where the goal embedding is concatenated with the concatenation of object representations and 2) *flat-attention (FA)* where the gated attention mechanism is applied at the scene-level rather than at the object-level (see Fig 1 in Appendix). *MA* significantly outperforms competing architectures on both sets. Appendix Section F provides detailed generalization performances organized by types of generalizations.



(a) (b)

Figure 2: **Reward function and policy learning.** a: Training (left) and testing (right) performances of the reward function after convergence (stars indicate significant differences w.r.t. *MA*. b: Training (plain) and testing (dashed) performances of the policy. *MA* outperforms *FA* and *FC* on both sets from $ep = 600$ ($p < 2 \cdot 10^{-3}$).

4.2 Robustness to Addition of Objects at Test Time

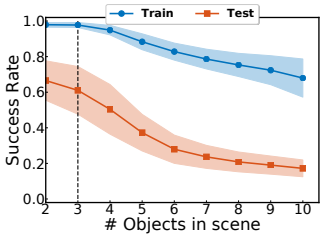


Figure 3: **Varying number of objects at test time.**

Fully-connected networks using concatenations of object representations are restricted to a constant number of objects (N). In contrast, *MA* architectures treat each object indifferently and in parallel which allows to vary N. Whether the performance of the architecture will be affected by N depends on the integration of object-specific information (*OR* for the reward function, *sum* and final network in the policy). Because the OR module is equivalent to a *max* function, it is not affected by N (given a perfect OR). The sum operator merges object-specific information to be used as input of a final network computing the action. As the sum varies with N, the overall policy will be sensitive to variations in N. Figure 3 shows the average training and testing performances of the policy as a function of N. We see that a model trained in scenes with $N = 3$ objects manages to maintain a reasonable performance on the training set for up to $N = 10$, while the generalization performance drops quickly as N increases. N could however be varied during training to make agent robust to its variations.

4.3 Introducing Two-Object Interactions

One can be concerned that the model presented above is limited to object-specific goals. As each module of the reward function receives as input observations from the agent's body and a single object, it cannot integrate multiple-object relationships to estimate the corresponding reward. In this section, we propose to extend the architecture to allow up to two-object relationships. Each module now receives observations from a pair of objects. For N objects, this results in $\binom{N}{2}$ modules (e.g. 6 for 4 objects). This way, each module is responsible for classifying whether its input pairs verifies the goal or not, while a logical OR integrates this two-object decisions into the global reward.

	1 obj	2 objs
Train	0.98 ± 0.01	0.92 ± 0.02
Test	0.94 ± 0.04	0.97 ± 0.02

Table 1: **$\mathbf{F_1}$-scores on one- and two-object goals.**

To test this, we reuse the dataset described in Section C and relabel its trajectories with one- and two-object goals related to the *grasp* predicate. More specifically, we add goals of the form *grasp* + *any* + ***relative position*** + ***color*** $\cup$ ***object type*** $\cup$ ***object category*** + *thing*, where ***relative position*** is one of $\{$*right_of, left_of, above, below*$\}$. For instance *grasp any right_of dog thing* is verified whenever the agent grasps an object that was *initially* at the right of any *dog*. These types of goals require to consider two objects: the object to be grasped and the reference object (*dog*). Table 1 shows that the reward function can easily be extended to considers object relations. Section G presents a description of the testing set and detailed performances by goal types.

5 Discussion

In this paper, we investigated how modular architectures of reward function and policy that operate on unordered sets of object-specific features could benefit generalization. In the context of language-guided autonomous exploration, we showed that the proposed architectures lead to both more efficient learning of behaviors from a training set and improved generalization on a testing set of goals. In addition we investigated generalization to different numbers of objects in the scene at test time and proposed an extension to consider goals related to object pairs.

Humans are known to encode persistent object-specific representations (Johnson, 2013; Green & Quilty-Dunn, 2017). Our modular architectures leverage such representations to facilitate transfer of knowledge and skills between object sharing common properties. Although these object features must presently be encoded by engineers, our architectures could be combined with unsupervised multi-object representation learning algorithms (Burgess et al., 2019; Greff et al., 2019).

Further work could provide agents the ability to select the number of objects in the scene, from which could emerge a curriculum: if the agent is guided by learning progress, it could first isolate specific objects and their properties, then generalize to more crowded scenes.

REFERENCES

Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in Neural Information Processing Systems*, pp. 5048–5058, 2017.

Dzmitry Bahdanau, Felix Hill, Jan Leike, Edward Hughes, Pushmeet Kohli, and Edward Grefenstette. Learning to Understand Goal Specifications by Modelling Reward. In *International Conference on Learning Representations*, jun 2019.

Peter W. Battaglia, Jessica B. Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, Caglar Gulcehre, Francis Song, Andrew Ballard, Justin Gilmer, George Dahl, Ashish Vaswani, Kelsey Allen, Charles Nash, Victoria Langston, Chris Dyer, Nicolas Heess, Daan Wierstra, Pushmeet Kohli, Matt Botvinick, Oriol Vinyals, Yujia Li, and Razvan Pascanu. Relational inductive biases, deep learning, and graph networks, 2018.

Benjamin Bergen. Embodiment, simulation and meaning. *The Routledge handbook of semantics*, pp. 142–157, 2015.

Christopher P Burgess, Loic Matthey, Nicholas Watters, Rishabh Kabra, Irina Higgins, Matt Botvinick, and Alexander Lerchner. Monet: Unsupervised scene decomposition and representation. *arXiv preprint arXiv:1901.11390*, 2019.

Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-attention architectures for task-oriented language grounding, 2017.

Cédric Colas, Pierre-Yves Oudeyer, Olivier Sigaud, Pierre Fournier, and Mohamed Chetouani. CURIOUS: intrinsically motivated modular multi-goal reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pp. 1331–1340, 2019a.

Cédric Colas, Olivier Sigaud, and Pierre-Yves Oudeyer. A hitchhiker's guide to statistical comparisons of reinforcement learning algorithms. *arXiv preprint arXiv:1904.06979*, 2019b.

Cédric Colas, Tristan Karch, Nicolas Lair, Jean-Michel Dussoux, Clément Moulin-Frier, Peter Ford Dominey, and Pierre-Yves Oudeyer. Language as a cognitive tool to imagine goals in curiosity-driven exploration. 2020.

Justin Fu, Anoop Korattikara, Sergey Levine, and Sergio Guadarrama. From Language to Goals: Inverse Reinforcement Learning for Vision-Based Instruction Following. In *International Conference on Learning Representations*, 2019.

Edwin James Green and Jake Quilty-Dunn. What is an object file? *The British Journal for the Philosophy of Science*, 2017.

Klaus Greff, Raphaël Lopez Kaufmann, Rishab Kabra, Nick Watters, Chris Burgess, Daniel Zoran, Loic Matthey, Matthew Botvinick, and Alexander Lerchner. Multi-object representation learning with iterative variational inference. *arXiv preprint arXiv:1903.00450*, 2019.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8): 1735–1780, November 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017. doi: 10.1109/cvpr.2017.215. URL http://dx.doi.org/10.1109/CVPR.2017.215.

Scott P Johnson. Object perception. *Handbook of developmental psychology*, pp. 371–379, 2013.

Michael P Kaschak and Arthur M Glenberg. Constructing meaning: The role of affordances and grammatical constructions in sentence comprehension. *Journal of memory and language*, 43(3): 508–529, 2000.

Daniel J Mankowitz, Augustin Žídek, André Barreto, Dan Horgan, Matteo Hessel, John Quan, Junhyuk Oh, Hado van Hasselt, David Silver, and Tom Schaul. Unicorn: Continual learning with a universal, off-policy agent. *arXiv preprint arXiv:1802.08294*, 2018.

Adam Santoro, David Raposo, David G. T. Barrett, Mateusz Malinowski, Razvan Pascanu, Peter W. Battaglia, and Timothy P. Lillicrap. A simple neural network module for relational reasoning. *CoRR*, abs/1706.01427, 2017. URL `http://arxiv.org/abs/1706.01427`.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Edward Wagstaff, Fabian B Fuchs, Martin Engelcke, Ingmar Posner, and Michael Osborne. On the limitations of representing functions on sets. *arXiv preprint arXiv:1901.09006*, 2019.

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pp. 3391–3401, 2017.

Vinicius Zambaldi, David Raposo, Adam Santoro, Victor Bapst, Yujia Li, Igor Babuschkin, Karl Tuyls, David Reichert, Timothy Lillicrap, Edward Lockhart, Murray Shanahan, Victoria Langston, Razvan Pascanu, Matthew Botvinick, Oriol Vinyals, and Peter Battaglia. Relational deep reinforcement learning, 2018.

A Environment and Grammar

A.1 Environment

The *Playground* environment is a continuous 2D world. In each episode, $N = 3$ objects are uniformly sampled from a set of 32 different object types (*e.g. dog, cactus, sofa, water, etc.*), organized into 5 categories (*animals, furniture, plants, etc.*), see Fig. 5. Sampled objects have a color (R,G,B) and can be grasped. Animals and plants can be grown when the right supplies are brought to them (food or water for animal, water for plants), whereas furniture cannot (e.g. sofa). Random scene generations are conditioned by the goals selected by the agent (e.g. *grasp red lion* requires the presence of a *red lion*).

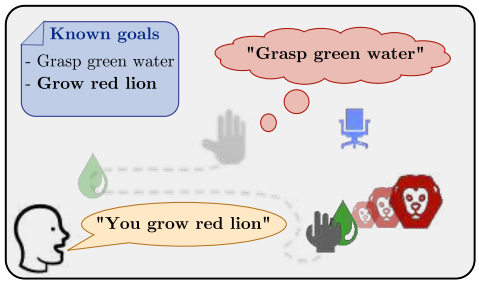


Figure 4: **The *Playground* environment.** The agent targets a goal represented as NL and receives descriptive feedback from the SP to expand its repertoire of known goals

Agent embodiment - In this environment, the agent can perform bounded continuous translations in the 2D plane, grasp and release objects by changing the state of its gripper. It perceives the world from an allocentric perspective and thus has access to the whole scene.

Agent perception - The scene is described by a state vector containing information about the agent's body and the N objects. Each object is represented by a set of features describing its type (one-hot encoding of size 32), its 2D-position, color (RGB code), size (scalar) and whether it is grasped (boolean). Categories are not explicitly encoded. Color, size and initial positions are sampled from uniform distributions making each object unique. At time step t, we can define an observation $\mathbf{o}_t$ as the concatenation of body observations (2D-position, gripper state) and objects' features. The state $\mathbf{s}_t$ used as input of the models is the concatenation of $\mathbf{o}_t$ and $\mathbf{\Delta o}_t = \mathbf{o}_t - \mathbf{o}_0$.

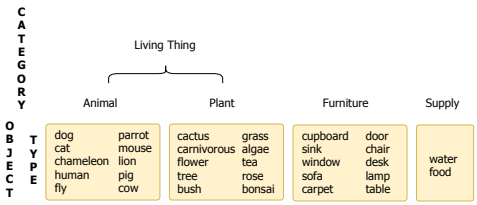


Figure 5: **Representation of possible objects types and categories.**

Social partner - Part of the environment, SP is implemented by a hard-coded function taking the final state of an episode ($\mathbf{s}_T$) as input and returning NL descriptions of $\mathbf{s}_T$: $\mathcal{D}_{\text{SP}}(\mathbf{s}_T) \subset \mathcal{D}^{\text{SP}}$. When SP provides *descriptions*, the agent hears targetable *goals*. Given the set of previously discovered goals ($\mathcal{G}_d$) and new descriptions $\mathcal{D}_{\text{SP}}(\mathbf{s}_T)$, the agent infers the set of goals that were not achieved: $\mathcal{G}_{na}(\mathbf{s}_T) = \mathcal{G}_d \setminus \mathcal{D}_{SP}(\mathbf{s}_T)$, where $\setminus$ indicates complement.

A.2 Grammar

1. Go: *(e.g. go bottom left)*
   - *go* + ***zone***
2. Grasp: *(e.g. grasp red cat)*
   - *grasp* + ***color*** $\cup$ {*any*} + ***object type*** $\cup$ ***object category***
   - *grasp* + *any* + ***color*** + *thing*
3. Grow: *(e.g. grow green animal)*
   - *grow* + ***color*** $\cup$ {*any*} + ***living thing*** $\cup$ {*living_thing, animal, plant*}
   - *grow* + *any* + ***color*** + *thing*

zone includes words referring to areas of the scene (e.g. *top*, *right*, *bottom left*), ***object type*** is one of 32 object types (e.g. *parrot*, *cactus*) and ***object category*** one of 5 object categories (*living_thing, animal, plant, furniture, supply*). ***living thing*** refers to any plant or animal word, ***color*** is one of *blue, green, red* and *any* refers to any color, or any object.

B 5 TYPES OF GENERALIZATION

We define 5 different types of *out-of-distribution* generalization:

- Type 1 - *Attribute-object generalization*: ***predicate*** + {*blue door, red tree, green dog*}. Understanding *grasp red tree* requires to leverage knowledge about the *red* attribute (grasping red non-tree objects) and the *tree* object type (grasping non-red tree objects).
- Type 2 - *Attribute extrapolation*: ***predicate*** + ***color*** ∪ {*any*} + *flower*. As *flower* is removed from the training set, *grasp red flower* requires the extrapolation of the red attribute to a new object type.
- Type 3 - *Predicate-category generalization*: *grasp* + ***color*** ∪ {*any*} + *animal*. Understanding *grasp any animal* requires to understand the animal category (from growing *animal* and growing animal objects) and the *grasp* predicate (from grasping non-*animal* objects) to transfer the former to the latter.
- Type 4 - *Easy predicate-object generalization*: *grasp* + ***color*** ∪ {*any*} + {*fly*}. Understanding *grasp any fly* requires to leverage knowledge about the *grasp* predicate (grasping non-fly objects) and the *fly* object (growing flies).
- Type 5 - *Hard predicate-object generalization*: *grow* + ***color*** ∪ {*any*} + ***plant*** ∪ {*plant, living_thing*}. *grow any plant* requires to understand the *grow* predicate (from growing animals) and the ***plant*** objects (and category) (from grasping plant objects). However, this transfer is more complex than the reverse transfer in Type 4 for two reasons. First, the interaction modalities vary: plants only grow with *water*. Second, Type 4 is only about the *fly* object, while here it is about all ***plant*** objects and the *plant* and *living_thing* categories.

Type 1	*Grasp blue door*, *Grasp green dog*, *Grasp red tree*, *Grow green dog*
Type 2	*Grasp any flower*, *Grasp blue flower*, *Grasp green flower*, *Grasp red flower*, *Grow any flower*, *Grow blue flower*, *Grow green flower*, *Grow red flower*,
Type 3	*Grasp any animal*, *Grasp blue animal*, *Grasp green animal*, *Grasp red animal*
Type 4	*Grasp any fly*, *Grasp blue fly*, *Grasp green fly*, *Grasp red fly*
Type 5	*Grow any algae*, *Grow any bonsai* *Grow any bush*, *Grow any cactus* *Grow any carnivorous*, *Grow any grass* *Grow any living_thing*, *Grow any plant* *Grow any rose*, *Grow any tea* *Grow any tree*, *Grow blue algae* *Grow blue bonsai*, *Grow blue bush* *Grow blue cactus*, *Grow blue carnivorous* *Grow blue grass*, *Grow blue living_thing* *Grow blue plant*, *Grow blue rose* *Grow blue tea*, *Grow blue tree* *Grow green algae*, *Grow green bonsai* *Grow green bush*, *Grow green cactus* *Grow green carnivorous*, *Grow green grass* *Grow green living_thing*, *Grow green plant* *Grow green rose*, *Grow green tea* *Grow green tree*, *Grow red algae* *Grow red bonsai*, *Grow red bush* *Grow red cactus*, *Grow red carnivorous* *Grow red grass*, *Grow red living_thing* *Grow red plant*, *Grow red rose* *Grow red tea*, *Grow red tree*

Table 2: Testing goals in $\mathcal{G}^{\text{test}}$

Each of the testing goals described above is removed from the training set ($\mathcal{G}^{\text{train}} \cap \mathcal{G}^{\text{test}} = \emptyset$). Table 2 provides the complete list of testing goals.

C DATASET

The reward function is trained in two contexts. First in a supervised setting, independently from the policy. Second, it is trained in parallel of the policy during RL runs. To learn a reward function in a supervised setting, we first collected a dataset of 50×10^3 trajectories and associated goal descriptions using a random action policy. Training the reward function on this data led to poor performances, as the number of positive examples remained low for some goals (see Fig. 6). To pursue the independent analysis of the reward function, we used 50×10^3 trajectories collected by an RL agent co-trained with its reward function using *modular-attention* architectures (data characterized by the top distribution in Fig. 6). Results presented in Fig. 2a used such RL-collected data. To closely match the training conditions imposed by the co-learning setting, we train the reward function on the final states s_T of each episode and test it on any state s_t for $t = [1, ..., T]$ of other episodes. The performance of the reward function are crucial to jointly learn the policy.

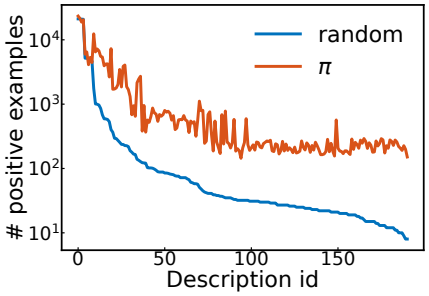


Figure 6: **Data distributions for the supervised learning of the reward function.** Sorted counts of positive examples per training set descriptions.

D ARCHITECTURE

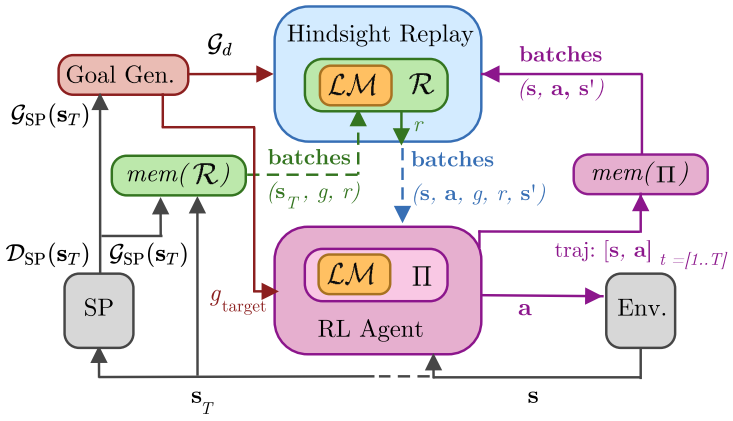


Figure 7: **The IMAGINE architecture.** Colored boxes represent the different modules composing IMAGINE. Lines represent update signals (dashed) and function outputs (plain). $\mathcal{LM}$ is shared.

Figure 7 represents the IMAGINE architecture whose logic can be outlined as follows:

1. The *Goal Generator* samples a target goal g_{target} from discovered goals ($\mathcal{G}_d$).
2. The agent interacts with the environment (*RL Agent*) using its policy Π conditioned by g_{target}.
3. The state-action trajectories are stored in *mem*(Π).
4. SP observes $\mathbf{s}_T$ and provides descriptions $\mathcal{D}_{\text{SP}}(\mathbf{s}_T)$ that the agent turns into targetable goals $\mathcal{G}_{\text{SP}}(\mathbf{s}_T)$.
5. *mem*($\mathcal{R}$) stores positive pairs $(\mathbf{s}_T, \mathcal{G}_{\text{SP}}(\mathbf{s}_T))$ and infers negative pairs $(\mathbf{s}_T, \mathcal{G}_{\text{na}}(\mathbf{s}_T))$.
6. The agent then updates:
   - *Goal Generator*: $\mathcal{G}_d \leftarrow \mathcal{G}_d \cup \mathcal{G}_{\text{SP}}(\mathbf{s}_T)$.
   - *Language Model* ($\mathcal{LM}$) and *Reward Function* ($\mathcal{R}$): updated using data from *mem*($\mathcal{R}$).
   - *RL agent* (actor and critic): A batch of state-action transitions $(\mathbf{s}, \mathbf{a}, \mathbf{s}')$ is sampled from *mem*(Π). Then *Hindsight Replay* and $\mathcal{R}$ are used to select goals to train on and compute rewards $(\mathbf{s}, \mathbf{a}, \mathbf{s}, g_{\text{NL}}, r)$. Finally, the policy and critic are trained via RL.

Descriptions of the language model, reward function and policy can be found in the main article. Next paragraphs describe others modules. Further implementation details, training schedules and pseudo-code can be found in the companion paper (Colas et al., 2020).

Language model - The language model ($\mathcal{LM}$) embeds NL goals ($\mathcal{LM}(g_{\text{NL}}) : \mathcal{G}^{\text{NL}} \rightarrow \mathbb{R}^{100}$) using an `LSTM` (Hochreiter & Schmidhuber, 1997) trained jointly with the reward function (yellow in Fig. 1). The reward function, policy and critic become language-conditioned functions when coupled with $\mathcal{LM}$, which acts like a goal translator.

Modular Reward Function using Deep Sets - Learning a goal-conditioned reward function ($\mathcal{R}$) is framed as a binary classification. The reward function maps a state $\mathbf{s}$ and a goal embedding $\mathbf{g} = \mathcal{LM}(g_{\text{NL}})$ to a binary reward: $\mathcal{R}(\mathbf{s}, \mathbf{g}) : \mathcal{S} \times \mathbb{R}^{100} \rightarrow \{0, 1\}$ (left in Fig. 1).

Architecture - The reward function, policy and critic leverage modular architectures inspired by Deep Sets (Zaheer et al., 2017) combined with gated attention mechanisms (Chaplot et al., 2017). Deep Sets is a network architecture implementing set functions (input permutation invariance). Each input is mapped separately to some (usually high-dimensional (Wagstaff et al., 2019)) latent space using a shared network. These latent representations are then passed through a permutation-invariant function (e.g. mean, sum) to ensure the permutation-invariance of the whole function. In the case of our reward function, inputs are grouped into object-dependent sub-states $\mathbf{s}_{obj(i)}$, each mapped to a probability p_i by a same network $\texttt{NN}^{\mathcal{R}}$ (weight sharing). $\texttt{NN}^{\mathcal{R}}$ can be thought of as a single-object reward function which estimates whether object i verifies the goal ($p_i > 0.5$) or not. Probabilities $[p_i]_{i \in [1..N]}$ for the N objects are then mapped into a global binary reward using a logical

OR function: *if any object verifies the goal, then the whole scene verifies it*. This OR function implements object-wise permutation-invariance. In addition to object-dependent inputs, the computation of p_i integrates goal information through a gated-attention mechanism. Instead of being concatenated, the goal embedding $\mathbf{g}$ is cast into an attention vector $\boldsymbol{\alpha}^g$ before being combined to the object-dependent sub-state through an Hadamard product (term-by-term) to form the inputs of $\texttt{NN}^{\mathcal{R}}$: $\mathbf{x}_i^g = \mathbf{s}_{obj(i)} \odot \boldsymbol{\alpha}^g$. This can be seen as scaling object-specific features according to the interpretation of the goal g_{NL}. Finally, we pre-trained a neural-network-based OR function: $\texttt{NN}^{\text{OR}}$ such that the output is 1 whenever $\max_i([p_i]_{i\in[1..N]}) > 0.5$. This is required to enable end-to-end training of $\mathcal{LM}$ and $\mathcal{R}$. The overall function can be expressed by:

$$\mathcal{R}(\mathbf{s}, g) \;=\; \texttt{NN}^{\text{OR}}([\texttt{NN}^{\mathcal{R}}(\mathbf{s}_{obj(i)} \odot \boldsymbol{\alpha}^g)]_{i\in[1..N]}).$$

We call this architecture *MA* for *modular-attention*.

Data - Interacting with the environment and SP, the agent builds a dataset of triplets $(\mathbf{s}_T, \mathbf{g}, r)$ where r is a binary reward marking the achievement of $\mathbf{g}$ in state $\mathbf{s}_T$. $\mathcal{LM}$ and $\mathcal{R}$ are periodically updated by backpropagation on this dataset.

Modular Policy using Deep Sets - Our agent is controlled by a goal-conditioned policy Π that leverages an adapted *modular-attention (MA)* architecture (right in Fig. 1). Similarly, the goal embedding $\mathbf{g}$ is cast into an attention vector $\boldsymbol{\beta}^g$ and combined with the object-dependent sub-state $\mathbf{s}_{obj(i)}$ through a gated-attention mechanism. As usually done with Deep Sets, these inputs are projected into a high-dimensional latent space (of size $N \times dim(\mathbf{s}_{obj(i)})$) using a shared network $\texttt{NN}^{\Pi}$ before being summed. The result is finally mapped into an action vector $\mathbf{a}$ with $\texttt{NN}^{\text{a}}$. Following the same architecture, the critic computes the action-value of the current state-action pair given the current goal with $\texttt{NN}^{\text{av}}$:

$$\Pi(\mathbf{s}, g) \;=\; \texttt{NN}^{\text{a}}\Big(\sum_{i\in[1..N]} \texttt{NN}^{\Pi}(\mathbf{s}_{obj(i)} \odot \boldsymbol{\beta}^g)\Big), \qquad Q(\mathbf{s}, \mathbf{a}, g) \;=\; \texttt{NN}^{\text{av}}\Big(\sum_{i\in[1..N]} \texttt{NN}^{Q}([\mathbf{s}_{obj(i)}, \mathbf{a}] \odot \boldsymbol{\gamma}^g)\Big).$$

Hindsight learning - Our agent uses *hindsight learning*, which means it can *replay* the memory of a trajectory (e.g. when trying to grasp object A) by *pretending* it was targeting a different goal (e.g. grasping object B) (Andrychowicz et al., 2017; Mankowitz et al., 2018; Colas et al., 2019a). In practice, goals originally targeted during data collection are replaced by others in the batch of transitions used for RL updates, a technique known as *hindsight replay* (Andrychowicz et al., 2017). To generate candidate substitute goals, we use the reward function to scan a list of 50 goals sampled randomly so as to bias the ratio of positive examples.

Goal generator - Generated goals are used to serve as targets during environment interactions and as substitute goals for hindsight replay. The goal generator samples uniformly from the set of discovered goals $\mathcal{G}_d$.

E COMPETING ARCHITECTURES

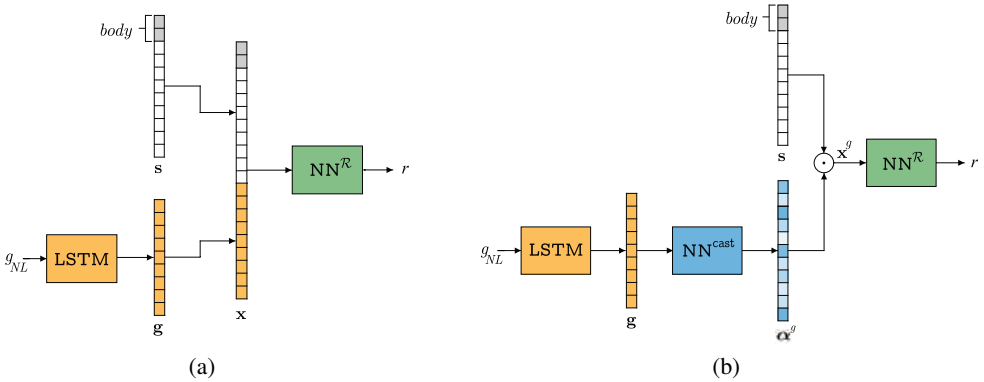
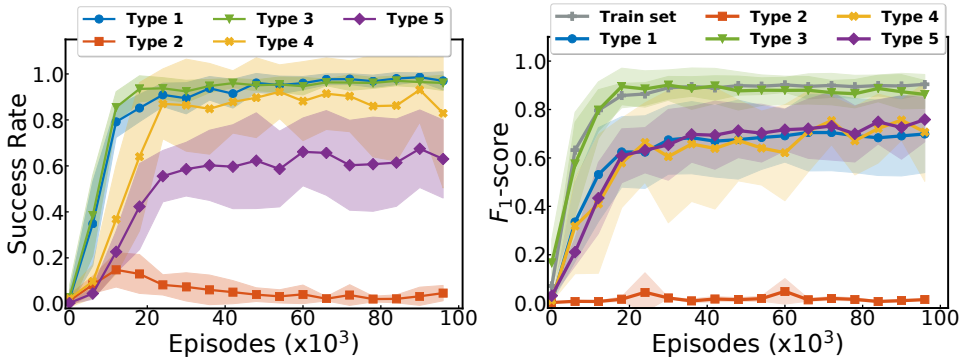


Figure 8: **Competing architectures.** a: Flat-concatenation (FC). b: Flat-attention (FA).

F RESULTS: GENERALIZATION PER TYPE

Fig. 9a provides the average success rate for the five generalization types. *MA* models demonstrate good generalizations of Type 1 (*attribute-object generalization*, e.g. *grasp red tree*), Type 3 (*predicate-category generalization*, e.g. *grasp any animal*) and Type 4 (*easy predicate-object generalization*: e.g. *grasp any fly*). Generalizing the meaning *grow* to other objects (Type 5, *hard predicate-object generalization*) is harder as it requires to understand the dynamics of the environment. As we could expect, the generalization of colors to new objects fails (Type 2, *attribute extrapolation*). As Type 2 introduces a new word, the language model's `LSTM` receives a new token, which perturbs the encoding of the sentence. The generalization capabilities of the reward function when it is jointly trained with the policy are provided in Fig. 9b. They seem to be inferior to the policy's capabilities, especially for Type 1 and 4. It should however be noted that the F_1-score plotted in Fig. 9b does not necessarily describe the actual generalization that occurs during the joint training of the reward function and the policy as it is computed from the supervised learning trajectories (see Section C).



(a) (b)

Figure 9: **Policy and reward function generalization.** a: Average success rate of the policy. b: F_1 score of the reward function.

G TWO-OBJECT RESULTS

Fig. 10 shows the evolution of the F_1-score of the reward function computed from the training set and the testing set (given in Fig. 11). The model considering two-objects interactions exhibit near perfect F_1-score for both one-object goals and two-objects goals. Note that, after convergence, the testing F_1-score is higher than the training one for two-objects goals. This is due to the fact that the testing set for two-objects goals is limited to only two examples.

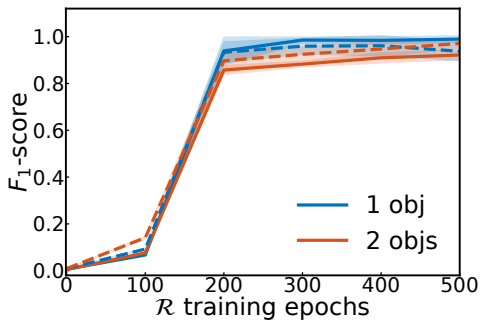


Figure 10: Convergence plot of the reward function. F_1-score w.r.t training epochs computed over the training (plain) and testing (dashed) sets for one-object goals (blue) and two-objects goals (red).

1 obj	*Grasp any animal*, *Grasp blue animal*, *Grasp red animal*, *Grasp green animal* , *Grasp any fly*, *Grasp blue fly* *Grasp red fly*, *Grasp green fly*, *Grasp blue door*, *Grasp green dog*, *Grasp red tree*
2 objs	*Grasp any left_of blue thing*, *Grasp any right_of dog thing*

Figure 11: Test goals used for the object-pair analysis.