AGENT, DO YOU SEE IT NOW? SYSTEMATIC GENERALISATION IN DEEP REINFORCEMENT LEARNING

Borja G. León[1] **& Murray Shanahan**[1] **& Francesco Belardinelli** [1,2]
[1] Department of Computing, Imperial College London, London, United Kingdom
[2] IBISC, Université d'Evry, Evry, France
{borjagleon, m.shanahan, francesco.belardinelli} @*imperial.ac.uk*

ABSTRACT

Systematic generalisation, i.e., the algebraic capacity to understand and execute unseen tasks by combining already known primitives, is one of the most desirable features for a computational model. Good adaptation to novel tasks in open-ended settings relies heavily on the ability of agents to reuse their past experience and recombine meaningful learning pieces to tackle new goals. In this work, we analyse how the architecture of convolutional layers impacts on the performance of autonomous agents when generalising to zero-shot, unseen tasks while executing human instructions. Our findings suggest that the convolutional architecture that is correctly suited to the environment the agent will interact with, may be of greater importance than having a generic convolutional network trained in the given environment.

1 INTRODUCTION

A major long-term goal of artificial intelligence research is the creation of generalist agents than can handle a never-ending stream of challenges. Open-ended research (Wang et al., 2019; 2020; Campero et al., 2021; Team et al., 2021) conceives algorithms that continuously improve themselves by creating both challenges and solutions. However, for this kind of algorithms to find meaningful solutions there is major concern on the generation of effective curriculum learning procedures that balance the right amount of optimization and exploration, facilitating generalisation. To achieve this, a relevant topic in open-ended research is investigating the drivers of generalisation in autonomous agents.

Systematic generalisation, also known as compositional generalisation (Lake, 2019), concerns the human capacity for compositional learning, that is, the algebraic capacity to understand and execute novel tasks by combining already known primitives. For instance, once a human agent understands the instruction "get *wood*" and the meaning of *iron*, they will understand the task "get *iron*". This is a desirable feature for a computational model, as it suggests that once the model is able to understand the components of a task, it should be able to satisfy tasks with the same components in possibly different combinations. The ability of neural-based agents to learn systematically and generalise beyond the training environment is a recurrent point of debate in machine learning (ML). In the context of autonomous agents solving human-given instructions, deep reinforcement learning (DRL) holds considerable promise (Co-Reyes et al., 2019; Luketina et al., 2019). Various research have shown that DRL agents can exhibit some degree of systematic generalisation with natural language (Yu et al., 2018; Bahdanau et al., 2019; Hill et al., 2021). However, it has become evident that systematic generalisation only *emerges* when the right environmental drivers are present (Mao et al., 2019; Lake, 2019), e.g. a fixed visual perspective.

One of the latest research in this area (Hill et al., 2020) focused on the emergence of generalisation with abstract logical operators. This work shows that DRL agents can generalise positive instructions from as few as six different specifications, while for negated instructions agents required a much larger variety of examples ($\sim$ 100). Consequently those findings suggest that when designing curriculum learning or open-ended algorithms for agents working with logical operators, e.g., "do *not* step into water", we should consider environments with larger varieties of instructions that we would when tackling positive instructions, e.g., "step into grass". Nevertheless, in this work we find that *systematically generalisation can emerge with instructions with logic operators such as negation as early as with specifications without them* and provide evidence that the architecture of the convolutional layers, i.e., the way that agents “see" their environment, plays a key role in the emergence of systematicity.

Contributions In this work we aim to answer the question: *can systematic learning emerge in DRL agents when prompted with abstract logic operators, e.g., disjunction, when learning from a limited variety ($\sim$ 6) of examples?* Specifically, we make the following contributions:

- We provide empirical evidence of emergent systematic generalisation in DRL agents with abstract operators – including negation, which previous works have struggled with – when learning from as few as six training instructions. Note that the previous latest research highlighted that significantly larger numbers ($\sim$ 100) were required (Hill et al., 2020).
- We find that the architecture of the convolutional layers (c.l.) – a background feature in previous literature – can be key when generalising to zero-shot tasks. Specifically, when variety of instructions is limited, we find that only those convolutional networks whose architecture is adapted to their environment generalise in a systematic fashion.

The remaining of this document is structured as follows: Section 2 introduces the key concepts needed to understand our research. Section 3 presents the formal language we use to provide instructions, while Section 4.2 specify the peculiarities of our DRL agents, focusing on the CNN architecture. Section 5 details the experimental settings and results obtained. Finally, Sections 6 and 7 report on related works and discussion, respectively.

2 Systematic generalisation of abstract operators

Our situated agents operate with a fixed visual perspective and limited visual range. Thus, we work with partially observable (p.o.) environments. Our target is to assess whether systematic learning is emerging in such environments while stressing that the convolutional neural network (CNN) architecture is key to facilitate generalisation when the number of training examples is limited. This section briefly introduces the concepts needed to present our work.

Reinforcement Learning. In RL p.o. environments are typically modelled as a POMDP.

Definition 1 (POMDP). *A* p.o. Markovian Reward Decision Process *is a tuple* $\mathcal{M} = \langle S, A, P, R, Z, O, \gamma \rangle$ *where (i)* S *is the set of states* $s, s', \ldots$ *(ii)* A *is the set of* actions $a, a', \ldots$ *(iii)* $P(s'|s,a) : S \times A \times S \to [0,1]$ *is the (probabilistic)* transition function. *(iv)* $R(s,a,s) : S \times A \times S \to \mathbb{R}$ *is the* Markovian reward function. *(v)* Z *is the set of* observations $z, z', \ldots$ *(vi)* $O(s,a,s) : S \times A \times S \to Z$ *is the* observation function. *(vii)* $\gamma \in [0,1)$ *is the* discount factor.

At every timestep t, the agent chooses an action $a_t \in A$ updating the current state from s_t to $s_{t+1} \in S$ according to P. Then, $R_t = R(s_t, a, s_{t+1})$ provides the reward associated with the transition, and $O(s_t, a_t, s_{t+1})$ generates an observation o_{t+1} that will be provided to the agent. Intuitively, the goal of the learning agent is to choose the policy π that maximizes the expected discounted reward from any state s. Given the action-value function $Q^{\pi}(s,a) = \mathbb{E}[R_t | s_t = s, a]$, RL searches a policy that maximizes the action-value function $Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a)$. For more detail we refer to Sutton & Barto (2018).

Convolutional neural network (CNN). A CNN is a specialized type of neural network model (LeCun et al., 2015) designed for working with visual data. Particularly, a CNN is a network with one or more convolutional layers (c.l.). Relevant features of c.l. for this study include *kernels*, a matrix of weights which is slid across an input image and multiplied so that the output is enhanced in a desirable manner, and *stride*, which measures the step-size of the kernel slid. After the last convolutional layer, the output is a tensor that we refer to as *visual embedding*. We point to Goodfellow et al. (2016) for a complete description of CNNs.

Evaluating the emergence of systematic generalisation. As anticipated in Sec. 1, we aim to assess whether a DRL agent is able to generalize systematically from instructions with abstract operators.

Consequently, *we do not focus on the raw reward collected by the agent*, but rather on whether the agent is correctly following new instructions. To evaluate this, we work in two popular settings involving procedural maps where agents need to fulfill reachability goals, in line with previous literature (Hill et al., 2020; Icarte et al., 2019; Kuo et al., 2020). In our experiments agents are trained with instructions such as "get gold" (reach gold) or "get something different from mud" (i.e., reach an object that is not mud). Then, we evaluate the agents with unseen instructions, e.g., "get something different from iron". It is evident that a learning agent achieving significantly better rewards than a random walker does not imply systematic learning, e.g., the agent could accumulate larger rewards just by navigating better than a random walker and reaching all the objects promptly. For this reason, after training, we evaluate generalisation in settings called binary choice maps (BCMs). In these test maps, agents are evaluated with zero-shot reliable instructions, i.e., unseen *reliable* instructions that point to objects granting a positive reward, or zero-shot *deceptive* instructions, i.e., instructions pointing to objects granting a penalisation – a negative reward –. Since during training we always provide reliable instructions, agents that show systematic learning – not complete systematicity but at least similar to closely related works (Bahdanau et al., 2019; Hill et al., 2020) – should accumulate significantly higher rewards with zero-shot reliable instructions than with deceptive ones (we estimate around two times higher or greater to be a good reference in our settings, detailed in Sec. 5).

3 Task Temporal Logic

Since we are concerned about generalisation with logic operators, we start by defining a language in temporal logic (TL) (Huth & Ryan, 2004) whose atomic components we use to train the DRL algorithm. Task temporal logic (TTL) is an expressive, learning-oriented TL language interpreted over finite traces, i.e, over finite episodes. Given a set AT of atomic tasks $\alpha, \beta, \ldots$, the syntax of TTL is defined as follows:

Definition 2 (TTL). *Every formula ϕ in TTL is built from atomic tasks $\alpha \in AT$ by using negation "$\sim$" (on atoms only), sequential composition ";" and non-deterministic choice of atoms "$\vee$" and formulae "$\cup$":*

$$\begin{aligned} T &::= \alpha \mid \alpha \sim \mid \alpha \vee \alpha \\ \phi &::= T \mid \phi; \phi \mid \phi \cup \phi \end{aligned}$$

Intuitively, an atomic task α corresponds to a reachability goal in the TL literature, in the sense that the fulfilment condition associated with α will eventually hold. Task $\alpha \sim$ encapsulates a special form of negation; informally it means that something different from α will eventually hold. Choosing this form of negation allows us to contrast our results with those of previous studies of this operator in natural language (Hill et al., 2020). Note that the negation symbol is intentionally positioned after the atomic task. We found that this feature helps during training with visual instructions, as it forces the neural module (Sec. 4.2) to process the same number of symbols to distinguish negative from positive tasks (we refer to Sec. E.1 in the supplemental material for further detail). Formulae $\phi; \phi'$ intuitively express that ϕ' follows ϕ in sequential order; whereas $\phi \cup \phi'$ means that either ϕ or ϕ' holds. We then also introduce an abbreviation for the concurrent (non-sequential) composition $\cap$ as follows: $\phi \cap \phi' \equiv (\phi; \phi') \cup (\phi'; \phi)$. We say that our language TTL is *learning-oriented* as its logical operators and their positions in formulae are so chosen as to help the training process of the learning agent.

TTL is interpreted over finite traces λ, where $|\lambda|$ denotes the length of the trace. We denote time steps, i.e., instants, on the trace as $\lambda[j]$, for $0 \leq j < |\lambda|$; whereas $\lambda[i, j]$ is the (sub)trace between instants i and j. In order to define the satisfaction relation for TTL, we associate with every atomic task α an atomic proposition p_α, which represents α's fulfilment condition. Note again that, in this context, α is a *reachability goal*, typically expressed in TL as an *eventuality* $\diamond p_\alpha$. Then, a *model* is a tuple $\mathcal{N} = \langle \mathcal{M}, L \rangle$, where $\mathcal{M}$ is a POMDP, and $L : S \to 2^{AP}$ is a labelling of states in S with truth evaluations of atoms p_α in some set AP of atoms.

Definition 3 (Satisfaction). *Let $\mathcal{N}$ be a model and λ a finite trace. We define the satisfaction relation $\models$ for tasks T and formulae ϕ on trace λ inductively as follows:*

$(\mathcal{N}, \lambda) \models \alpha$ *iff for some* $0 \leq j < |\lambda|$, $p_\alpha \in L(\lambda[j])$
$(\mathcal{N}, \lambda) \models \alpha \sim$ *iff for some* $0 \leq j < |\lambda|$, *for some* $q \neq p_\alpha$, $q \in L(\lambda[j])$ *and* $p_\alpha \notin L(\lambda[j])$

$$\begin{array}{lll}
(\mathcal{N}, \lambda) \models \alpha \vee \alpha' & \textit{iff} & (\mathcal{N}, \lambda) \models \alpha \textit{ or } (\mathcal{N}, \lambda) \models \alpha' \\
(\mathcal{N}, \lambda) \models \phi; \phi' & \textit{iff} & \textit{for some } 0 \leq j < |\lambda|, (\mathcal{N}, \lambda[0, j]) \\
 & & \models \phi \textit{ and } (\mathcal{N}, \lambda[j+1, |\lambda - 1|]) \models \\
 & & \phi' \\
(\mathcal{N}, \lambda) \models \phi \cup \phi' & \textit{iff} & (\mathcal{N}, \lambda) \models \phi \textit{ or } (\mathcal{N}, \lambda) \models \phi'
\end{array}$$

By Def. 3, an atomic task α indeed corresponds to a formula $\diamond p_\alpha$ of temporal logic, where p_α is the fulfilment condition associated with α. It can be immediately inferred that TTL is a fragment of the widely-used Linear-time Temporal Logic over finite traces (LTL$_f$) (De Giacomo & Vardi, 2013). We provide a translation of TTL into LTL$_f$ and the corresponding proof of truth preservation in Sec. F in the appendix. To better understand the language we present a toy example.

Example 1. *The specification "get something different from grass, then collect grass and later use either the workbench or the toolshed" can be expressed in TTL as:*

$$\phi \triangleq (grass \sim); grass; (workbench \cup toolshed)$$

TTL is designed to be expressive enough to encapsulate both the tasks described in (Andreas et al., 2017), a popular benchmark in the RL-TL literature (León & Belardinelli, 2020; Toro Icarte et al., 2018; De Giacomo et al., 2019), and the negated instructions from studies about negation in natural language (Hill et al., 2020), i.e., encapsulating "not wood" as "get something different from wood".

Remark. *As a foundational work assessing the ability of learning agents to generalise abstract operators with unseen instructions, we intentionally restricted TTL syntax with atomic operators to negation and disjunction. This allows to carefully study the ability of DRL agents to generalise with these two important operators.*

Given a task T and an event detector $\mathcal{L}_I : Z \to 2^{AP}$ that maps observations into sets of atoms in AP. The reward function for the DRL agent is defined as follows:

$$R(p_t) = \begin{cases} d & \text{if } p_t = \emptyset, \text{ where } d \in \mathbb{R} \text{ and } d < 0; \\ i & \text{if } p_t = p_\alpha, \text{ for } \alpha \text{ occurring in } T \text{ and} \\ & T \neq \alpha \sim, \text{ or if } p_t \neq p_\alpha, \text{ for } \alpha \text{ occurring in} \\ & T \text{ and } T \equiv \alpha \sim, \text{ where } i \in \mathbb{R} \text{ and } i \geq 0; \\ c & \text{otherwise, where } c \in \mathbb{R} \text{ and } c \ll d. \end{cases}$$

where $\mathcal{L}_I(z_t) = p_t$ for any time step t. Intuitively, the agent is rewarded when the event detector $\mathcal{L}_I$ fires a signal that satisfies the current task, and penalised when $\mathcal{L}_I$ signals an event not related to T. A small penalisation is also given at each time step to induce the agent to promptly solve T.

4 Generalising TTL formulae

In a similar fashion to previous literature applying temporal logic to RL (TL-RL), we rely on a neuro-symbolic framework composed by a symbolic module (SM) and a neural module (NM), the former specializes in generalising formulae operators while the later is a DRL algorithm that learns atomic operators in a systematic fashion. Note that our main contribution is the analysis of the impact of different CNN architectures in DRL agents. Nevertheless, as bridging work between RL-TL and systematic generalisation literature, we detail both modules in the section below to provide a complete picture of our method.

4.1 Symbolic Module

Given an instruction ϕ in TTL, the first task of the symbolic module (SM) (detailed in Algorithm 1 in Appendix B), is to decompose ϕ into sequences of atomic tasks for the neural module (NM). Intuitively, the SM reduces the problem into a progression of POMDPs, which is a well-known procedure in the literature on RL with TL specifications (Brafman et al., 2018; Toro Icarte et al., 2018). In particular, a task extractor $\mathcal{E}$ transforms ϕ into the set $\mathcal{K}$ of all sequences of tasks T that satisfy ϕ (see Algorithm 2). As standard in the literature (Andreas et al., 2017; Kuo et al., 2020), we assume that the SM has access to an internal labelling function (a.k.a. event detector[1]) $\mathcal{L}_I : Z \to 2^{AP}$, which maps the agent's observations into sets of atoms in AP. Note that $\mathcal{L}_I$ might differ from the labelling $\mathcal{L}$ originally defined in Sec. 3, since $\mathcal{L}$ is defined on states. For our approach to be effective, observations have to contain enough information to decide the truth of atoms, so that both the agent's internal and the model's overall labelling functions remain aligned. Given $\mathcal{L}_I$ and the sequences of tasks in $\mathcal{K}$, a progression function $\mathcal{P}$ selects the next task from $\mathcal{K}$ that the NM has to solve. Once a task is solved, $\mathcal{P}$ updates $\mathcal{K}$ and selects the next task. This process is repeated until ϕ is satisfied or the time limit is reached. We include the pseudocode for $\mathcal{K}$ and $\mathcal{P}$ subroutines in Algorithms 2 and 3 of the Appendix respectively.

When forwarding the current task T to the NM, the SM expands observation z with the subformula corresponding to T. In the current context, z is a square visual input with a fixed perspective. In our empirical setting, this is expanded either by adding extra rows to z containing T (see Figure 1 center) or providing T in a separated "language" channel (see Figure 1 right). We now illustrate the inner working of the SM.

Example 2. *Consider one of the complex specifications used in Sec. 5: $\phi_1 = ((wood; grass) \cup (iron; axe)); workbench; (toolshed\ \sim)$. Given ϕ_1, the task extractor $\mathcal{E}$ outputs set $\mathcal{K} = \{\langle [wood], [grass], [workbench], [toolshed\ \sim] \rangle, \langle [iron], [axe], [workbench], [toolshed\ \sim] \rangle\}$ with two available sequences of tasks. Set $\mathcal{K}$ is forwarded to the progression function $\mathcal{P}$ and since we have two lists with different initial elements, both positive, $\mathcal{P}$ selects $[wood \vee iron]$ as task T to extend the next observations of the NM. The NM iterates with the environment using the extended observations as input. At each iteration, functions $\mathcal{L}_I$ and R respectively evaluate the progression of T and reward the NM until T is satisfied. In our example, if the NM got wood, the second sequence and $[wood]$ are discarded and the next task becomes $[grass]$. This process is repeated until $\mathcal{K}$ is empty or the time limit is reached.*

Note that the main focus of this work is to assess generalisation with abstract operators such as negation and disjunction. Thus our experiments and results in the main document will focus on generalising atomic tasks T. For generalisation of TTL formulae we refer to Appendix B.

4.2 Deep RL Agents

We aim to demonstrate that DRL agents can generalise logic TTL operators from a limited variety of instructions and the major role that convolutional layers play in the ability of agents towards generalisation. In this section we detail the agents used in our experiments.

Common features. We use A2C, a synchronous version of the DRL algorithm A3C (Mnih et al., 2016). Agents are trained in procedurally generated maps, using a fixed perspective which is known to help agents with generalisation (Mao et al., 2019). The deep learning architecture is composed by a CNN, followed by a fully-connected (FC) layer and an LSTM (LeCun et al., 2015). When instructions are provided as text we use a bidirectional LSTM concatenate its output with the one from the CNN. Agents only differ in the architecture of their CNN. Further detail is given in Appendix A

Convolutional architectures. In preliminary studies, we observed that agents with different FC and recurrent layer configurations can exhibit similar generalisation abilities. However, as detailed in Sec. 5, different convolutional configurations yield significantly different results. Particularly, while all the convolutional architectures we test achieve a similarly good performance with training instructions, only specific configurations correctly execute zero-shot tasks. We find that the ability to generalise is correlated to the degree of *alignment* between the CNN architecture and the environment. In our gridworld settings, we say that a CNN is weakly aligned (WA) with an environment when the kernel dimensions of the c.l. and stride size are factors of the setting's tile resolution. If a CNN is WA and the two first dimensions (length and width) of its visual embedding correspond to the number of input tiles in the original observation, we say that this CNN is strongly aligned (SA). Note that the number of channels of the output, i.e., the third dimension of the embedding, does not influence the alignment. Intuitively, the key feature is that a SA CNN is one that produces an independent output for each object in the environment. Last, when a CNN is not WA or SA, we say it is not aligned (NA).

[1]Note that RL literature and vanilla RL settings (Sutton & Barto, 2018) also rely on these event detectors to provide rewards to the agents although they are not formally defined.

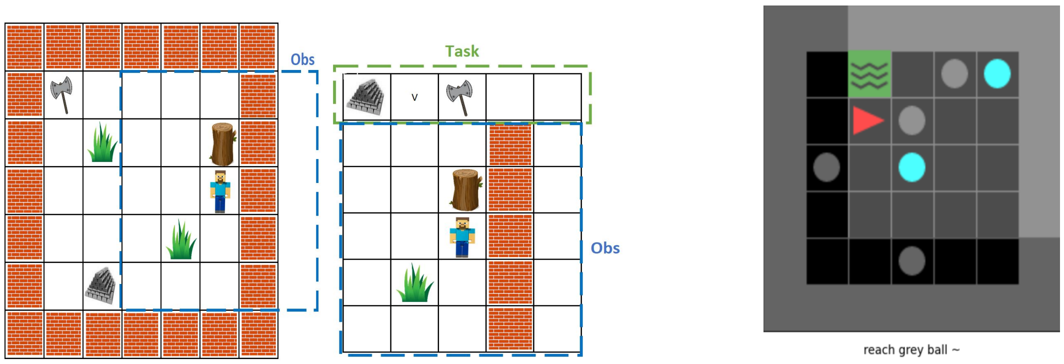

Figure 1: **Left**. Example of a procedurally generated test map (left) in the Minecraft-inspired environment with its corresponding extended observation (**center**) when the agent's task is to get either iron or an axe (top row). Tasks are specified in the top row of the extended observation in this setting by depicting the object itself. **Right**. A MiniGrid map where the task is to reach an object different from the grey ball. Here instructions are given on a separated channel and objects are combinations of shapes and colors.

When aiming compositional learning from ~ 6 training instructions, we find that only SA networks (when giving visual instructions) or SA and WA networks (when instructions are given in a separated channel) correctly execute zero-shot tasks.

5 Experiments

We evaluate our framework in procedurally generated grid-worlds. The first setting (Figure 1 left) is designed along the lines of a Minecraft-inspired (Minecraft for sort) benchmark widely used in RL-TL (Andreas et al., 2017; Toro Icarte et al., 2018; De Giacomo et al., 2019), which we adapt to match the 2D maps from Hill et al. (2020) (the last study about negation). The action set consists of 4 individual actions: *Up, Down, Left, Right*, that move the agent one tile in the given direction. In this setting each tile has a resolution of 9x9x1 values. The DRL algorithm receives the observation extended with visual instructions depicting the target objects (Figure 1 center). We also evaluate the agents in the popular MiniGrid benchmark (Chevalier-Boisvert et al., 2018), depicted in Figure 1 right. Here each tile has a resolution of 8x8x3 values, and the action set is *Move forward, Turn left, Turn right*. In this benchmark instructions are given through a separated language channel.

For each setting we test four different CNN configurations. The first two are popular architectures from previous literature that we use as baselines: *Atari-CNN* (Mnih et al., 2015; 2016) which is NA with Minecraft and WA with MiniGrid, and *ResNet* (Espeholt et al., 2018; Hill et al., 2020) WA with Minecraft and NA with MiniGrid. We contrast these networks with different architectures designed to be SA with each environment and that we refer to as Env-CNN. Last, for each environment we also test the performance of the same Env-CNNs, but when the c.l. are frozen during training, i.e., the c.l. always use a set of randomly selected parameters. We refer to these networks as Env-CNN$^{\mathrm{RF}}$. Schemes of the architectures are given in Appendix A.1.

In every experimental setting we refer to the global set of objects as $\mathcal{X}$. $\mathcal{X}$ is split into training ($\mathcal{X}_1$), validation ($\mathcal{X}_2$) and test sets ($\mathcal{X}_3$) where $\mathcal{X} = \mathcal{X}_1 \cup \mathcal{X}_2 \cup \mathcal{X}_3$, $\mathcal{X}_1 \cap \mathcal{X}_2 = \emptyset$, $\mathcal{X}_1 \cap \mathcal{X}_3 = \emptyset$, and $\mathcal{X}_2 \cap \mathcal{X}_3 = \emptyset$. We contrast the results of learning with training sets of different size that we refer to as *small* ($|\mathcal{X}_1| = 6$) and *large* ($|\mathcal{X}_1| = 20$). In Minecraft, where instructions are visual, agents are trained with objects from $\mathcal{X}_1$ only. In MiniGrid, where agents require to ground words to objects, we train the agents with positive instructions from $\mathcal{X}$, while instructions concerning the negation or disjunction operators refer only to objects in $\mathcal{X}_1$. Agents are trained with atomic tasks T. Tasks are procedurally generated with objects from the corresponding set. For instance, with the small setting we have six different objects, meaning that agents are learning the negation operator from six training instructions. At each episode the agent navigates receiving rewards according to T. The episode ends when the agent receives a positive reward (meaning that T is satisfied) or the time limit is reached. See Appendix B for the reward function. During training we periodically evaluate the performance of

Table 1: Mean and standard deviation from the average rewards obtained from five independent runs in a set of 500 maps with atomic tasks. A value of 1.0 is the performance of a random walker. The highest value over all networks is bolded.

Minecraft	Atari-CNN (**NA**)		ResNet (**WA**)		Env-CNN$_{\text{minec.}}$ (**SA**)		Env-CNN$^{\text{RF}}_{\text{minec.}}$ (**SA**)	
$\lvert\mathcal{X}_1\rvert$	Train	Test	Train	Test	Train	Test	Train	Test
6	3.0(0.1)	**2.9**(0.2)	3.2(0.49)	2.7 (0.22)	**4.5**(0.75)	1.8(0.3)	2.9(0.8)	2.6(0.8)
20	3.1(0.1)	3.1(0.1)	**4.5**(0.4)	2.1(0.5)	4.5(0.3)	**3.8**(0.7)	3.3(1.0)	3.1(0.9)
Minigrid	Atari-CNN (**WA**)		ResNet (**NA**)		Env-CNN$_{\text{minig.}}$ (**SA**)		Env-CNN$^{\text{RF}}_{\text{minig.}}$ (**SA**)	
$\lvert\mathcal{X}_1\rvert$	Train	Test	Train	Test	Train	Test	Train	Test
6	**4.0**(1.1)	**4.0**(0.9)	4.0(0.9)	3.6 (0.7)	2.8(1.03)	2.3(1.0)	3.3(0.8)	3.4(0.3)
20	6.2(0.8)	3.3(1.1)	5.6(0.8)	**5.6**(1.0)	**7.3**(0.5)	4.1(0.9)	4.7(0.2)	2.6(3.4)

Table 2: Results from 500 binary choice maps with zero-shot instructions. In 'Reliable' higher is better while in 'Deceptive' lower is better. Agents show a stronger systematic generalisation the bigger the difference between respective reliable and deceptive instructions. Positive and negative labels refer to atomic positive or negative instruction respectively pointing to one of the two objects present. In disjunction 2nd choice the instructions refers to two objects, but only the second is present in the map. Disjunction 2 choices points to two objects in a map with 4 items. Results are bolded where performance with reliable instructions is greater or equal than two times the respective performance with deceptive instructions.

Minecraft	$\lvert\mathcal{X}_1\rvert$	Atari-CNN (**NA**)		ResNet (**WA**)		Env-CNN$_{\text{minecraft}}$ (**SA**)		Env-CNN$^{\text{RF}}_{\text{minecraft}}$ (**SA**)	
Task		Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive
Positive	6	3.9(0.1)	3.9(0.1)	3.7(0.5)	3.4(0.8)	**3.3**(1.9)	**0.3**(1.9)	**4.0**(1.8)	**0.7**(0.7)
	20	4.1(0.1)	4.2(0.1)	**3.3**(1.8)	**1.1**(1.4)	**5.4**(0.2)	**0.1**(0.0)	**4.0**(1.8)	**1.1**(1.2)
Negative	6	3.8 (0.2)	3.8(0.1)	3.5(0.5)	3.5(0.7)	**2.2**(1.3)	**0.4**(0.4)	**2.2**(1.9)	**1.0**(0.8)
	20	3.9(0.1)	4.0(0.1)	**3.1**(1.7)	**1.3**(0.9)	**4.4**(0.7)	**0.1**(0.1)	**2.3**(1.8)	**0.9**(1.2)
Disj.	6	2.6 (0.2)	2.6(0.6)	2.3(0.1)	2.5(0.7)	**2.2**(4.8)	**0.6**(0.3)	2.5(0.7)	1.4(0.1)
2 choices	20	2.6 (0.2)	2.7(0.6)	2.2(0.1)	2.2(0.7)	**3.5**(4.8)	**0.4**(0.3)	**2.8**(0.7)	**1.1**(0.9)
Minigrid	$\lvert\mathcal{X}_1\rvert$	Atari-CNN (**WA**)		ResNet (**NA**)		Env-CNN$_{\text{minigrid}}$ (**SA**)		Env-CNN$^{\text{RF}}_{\text{minigrid}}$ (**SA**)	
Task		Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive
Negative	6	**2.8** (1.0)	**0.4**(0.3)	3.0(1.4)	3.0(1.4)	**2.0**(1.3)	**0.4**(0.3)	**1.6**(0.2)	**0.8**(0.2)
	20	**2.8**(0.8)	**0.3**(0.1)	**5.1**(1.3)	**0.2**(0.1)	**5.0**(1.7)	**0.2**(0.1)	**2.2**(0.2)	**1.0**(0.2)
Disj.	6	**4.2** (0.5)	**0.8** (0.5)	4.0(0.4)	3.68(0.6)	**2.7** (1.5)	**0.8**(0.4)	**4.1**(0.7)	**0.6**(0.4)
2 choices	20	**3.4**(0.97)	**1.1**(0.48)	**5.0**(0.6)	**1.8**(0.8)	**3.1**(0.5)	**0.4**(0.1)	2.7(1.0)	1.6(0.4)
Disj.	6	**2.8** (0.9)	**0.9**(0.4)	3.5(0.8)	3.9(1.3)	**1.7**(1.3)	**0.8**(0.3)	**2.8**(0.8)	**0.9**(0.6)
2nd choice	20	2.1(1.4)	1.1(0.3)	2.6(0.7)	2.4(1.0)	**1.8**(0.5)	**0.5**(0.1)	1.9(0.8)	1.3(0.4)

the agents with the corresponding validation set $\mathcal{X}_2$, that we use to select the best learning weights in each run, i.e., to perform early stopping when necessary. Further details about object set generation, reward values, environment mechanics and training and validation plots are included in Appendix A.

Results about systematic learning. Table 1 presents the raw reward results in $\mathcal{X}_1$ and $\mathcal{X}_3$ with the different CNNs. Note that "train" refers to results with objects from the training set but with the weights that achieve the best rewards in $\mathcal{X}_2$. Therefore, all the networks can accumulate higher rewards in training if we let them overfit (See Figure 3 in appendix for training and validation plots). Note that results in Table 1 alone do not yield a strong correlation between the alignment of the CNN and the generalisation ability. Yet, as anticipated in Sec. 2 this correlation becomes evident when evaluating whether the agents are actually following the test instructions. Table 2 shows the results in a particular set that we call binary choice maps (BCMs). Here maps have only two objects (except in "disjunction 2 choices", with four), one granting a positive reward ($i = +1$) the other a penalisation ($c = -1$). In "disjunction 2 choices" two objects penalise, and two provide a reward. Since agents are always trained with reliable instructions pointing to objects that grant a positive reward, an agent that is correctly generalising to zero-shot instructions should collect significantly higher rewards with reliable instructions than with deceptive ones (i.e., the later instructions are pointing to objects that grant a negative reward). Roughly, agents should accumulate with reliable instructions two times greater rewards than when given deceptive instructions to show similar generalisation to the experiments in previous studies about negation with 100 training instructions (Hill et al., 2020).

The first strong pattern that we extract from Table 2 is that only architectures with some form of alignment (WA and SA) give evidence of systematic learning when no more than six training instructions are available. NA architectures show minimal to none generalisation (e.g. better performance with deceptive instructions than with reliable) in all the small settings. With 20 objects, NA architectures also fail when instructions are visual (Minecraft) while struggling with the disjunction operator when instructions come in a dedicated channel (Minigrid). In the latter setting, architectures with some alignment (WA) show good generalisation performance. Yet, when instructions are visual they only learn to generalise positive and negative instructions, and exclusively in the larges setting. Notably, architectures that are strongly aligned (SA) show good generalisation even when the weights of the c.l. are not trained (Env-CNN$^{\text{RF}}$), further evidencing the high influence of the architecture configuration in how the agent generalises. Remarkably, the Env-CNN (the fully trained SA architecture) is the only configuration that systematically achieves a performance greater than 1 (the performance of a random agent) with reliable instructions while getting less than 1 with deceptive instructions across all the settings. That is, *SA architectures are the only configurations that have systematically learnt to generalise from the training instructions to guess not only the desired object/s, but also which undesired items should be avoided according to the test instruction.*

Last, we highlight that in preliminary studies we also explored additional overfitting-prevention methods, including dropout and autoencoders, to help NA networks. However, the results do not vary from the ones obtained by the train-validation-test set division that we carry here and that evidence the benefits of using architectures that are aligned with its environment. We included the disjunction test where agents struggled the most, due to space constrains, additional results are included in Appendix C.

6 Related work

Here we include the closest related literature to this research:

Systematic Generalisation. Generalisation beyond training instructions has been widely studied in the RL literature. For instance, Oh et al. (2017) presents a framework that relies on hierarchical RL and task decomposition of instructions in English to enhance generalisation in DRL. Later work from Mao et al. (2019) introduces a neuro-symbolic concept learner that jointly learns visual concepts, words, and semantic parsing of sentences from natural supervision. Yu et al. (2018) present a model that reports strong generalisation in a 2D environment that is aimed at language understanding for positive instructions. Closer to our line of work, there is growing literature (see e.g., Smolensky (1988); Bahdanau et al. (2019); Lake (2019)) focused on finding the right drivers enabling systematic learning from natural language instructions, e.g., fixed perspectives and varied training data. Recent work (Hill et al., 2020) suggests that systematic generalisation is not a binary question, but an *emergent* property of agents interacting with a situated environment (McClelland et al., 2010) and explores the drivers that enable agents generalise an abstract operator such as negation. We expand the state of this line of works by first, providing evidence that systematic learning can emerge with logic operators from as few examples as with positive instructions, second, giving evidence that the CNN architecture is a key driver –that previous research was oblivious about– towards generalisation.

Reinforcement Learning and Temporal Logic. Training autonomous agents to solve multi-task goals expressed in temporal logic is drawing growing attention from the research community. Examples include following instructions expressed as TL formulae by using logic progression and RL (Littman et al., 2017; Andreas et al., 2017), tackling continuous-action environments (Yuan et al., 2019), multi-agent systems (León & Belardinelli, 2020) and studies on the applicability of different TL languages (Camacho et al., 2019). Complex goals expressed as temporal formulae can also be easily decomposed by using other techniques, including finite state automata (De Giacomo et al., 2019; Icarte et al., 2018) and progression (Toro Icarte et al., 2018). More recent work has further exploited automata techniques, combining hierarchical approaches and reward machines – a particular form of automata– (Furelos-Blanco et al., 2021) or high-level planning with low-level RL solutions (Jothimurugan et al., 2021). These works focus on solving non-Markovian models by extending the state space in a minimal form (to minimally impact the state space) so that the RL agent learns from an equivalent Markovian model, where an optimal behavior is also optimal in the original system (Bacchus et al., 1996). A drawback of the mentioned literature is their exclusive focus on learning to solve compositions of training tasks. For this reason Kuo et al. (2020) targets generalisation of temporal logic instructions with DRL at the expense of requiring an additional neural network for each new symbol or object encountered. Our work advances the state of the art by presenting a framework that can execute zero-shot TL formulae while relying on a single neural network.

7 Discussion and conclusions

We have empirically demonstrated that systematic learning can emerge with abstract operators including negation as early as with positive instructions. An obvious limitation of our work is the use of regular gridworlds in our experiments. However, this feature does does not lessen our contributions since previous closely-related work have struggled to provide evidence of generalisation with abstract operators in similar settings. As we previously discussed in Sec. 1, the ability of deep learning agents for compositional reasoning is a recurrent topic in the literature. Our results suggest that future research should pay close attention to the CNN architecture employed in their benchmarks before calling into question whether it is possible for agents to generalise. Another limitation is that — in line with related work on generalisation (Lake & Baroni, 2018; Bahdanau et al., 2019; Hill et al., 2020) – we do not give nomological explanations for the generalisation ability (or lack thereof) that we evidence. Yet, this is beyond the scope of our contribution, which is proving the emergence of generalisation and the key elements missing in previous works in similar settings.

As a foundational bridging work between the literature of formal methods applied to RL (FM-RL) and systematic generalisation, we use a language that is less expressive than the latest contributions in the field. This constraint was needed to carry a careful experimental evaluation of the systematic learning with the different types of operators. Given the positive results, more expressive languages can be used in future iterations. Nevertheless, we presented the first framework that, leveraging on a learning-oriented syntax and the compositional structure of logical formulae, is capable of executing zero-shot complex formulae in temporal logic while relying on a single network. As anticipated in Sec. 6, this contrasts with dominant methods in the FM-RL literature that rely on a new policy network per task (e.g., De Giacomo et al. (2019); Icarte et al. (2019); Jothimurugan et al. (2021); Furelos-Blanco et al. (2021)) or a new network per word and symbol Kuo et al. (2020).

Our findings evidence that from as few as six different examples agents with aligned architectures learn not only how to follow training commands, but also abstract information about how symbols and operators in the given language compose and how the combination of those symbols influences what the agent should do. This contrasts with Hill et al. (2020) – the only previous study highlighting some generalisation with unseen negated instructions – which suggests that $\sim$100 instructions are required with negated instructions. Using the same form of negation, we find that the CNN architecture plays a key role when learning from limited examples. Still, we observe in our experiments that the ResNet used in Hill et al. (2020) shows evidence of generalising negation with 20 instructions. This also contrasts with Hill et al. (2020), where the ResNet performed barely better than random when trained with 40 negated instructions. We find that this difference comes from our use of a validation set, not common in RL literature and consequently not present in previous research. We provide empirical evidence of this point in Appendix D. This latest point also evidences the importance of adopting the traditional train-validation-test set split from supervised learning literature into RL research.

In the context of open-ended research, our results are also specially relevant as they highlight a key feature for the ability of agents to acquire new knowledge. Moreover, the good generalisation results with the randomnly-parameterized convolutional layers in both settings suggests that finding a suited architecture may be more beneficial than training a generic convolutional configuration when aiming open-ended agents. Consequently, when working in visually continuous settings – where the prior-knowledge about object resolution cannot be applied – our research indicates that methods for automatic generation of deep learning architectures are a potential way forward (Elsken et al., 2019; Song et al., 2020). The impact of making an explicit train-validation-test splits when assessing the RL agent ability is also of special importance for open-ended literature, where often environments are generated by learning agents Wang et al. (2019); Campero et al. (2021). Future works should assess performance in zero-shot environments in a similar fashion to the supervised learning literature LeCun et al. (2015).

REFERENCES

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 166–175. JMLR. org, 2017.

Fahiem Bacchus, Craig Boutilier, and Adam Grove. Rewarding behaviors. In *Proceedings of the National Conference on Artificial Intelligence*, pp. 1160–1167, 1996.

Dzmitry Bahdanau, Shikhar Murty, Michael Noukhovitch, Thien Huu Nguyen, Harm de Vries, and Aaron C. Courville. Systematic generalization: What is required and can it be learned? In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

Ronen I Brafman, Giuseppe De Giacomo, and Fabio Patrizi. Ltlf/ldlf non-markovian rewards. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Alberto Camacho, R Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Ltl and beyond: Formal languages for reward function specification in reinforcement learning. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 6065–6073, 2019.

Andres Campero, Roberta Raileanu, Heinrich Küttler, Joshua B Tenenbaum, Tim Rocktäschel, and Edward Grefenstette. Learning with amigo: Adversarially motivated intrinsic goals. In *International Conference on Learning Representations*, 2021.

Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. Minimalistic gridworld environment for openai gym. `https://github.com/maximecb/gym-minigrid`, 2018.

John D Co-Reyes, Abhishek Gupta, Suvansh Sanjeev, Nick Altieri, John DeNero, Pieter Abbeel, and Sergey Levine. Meta-learning language-guided policy learning. In *International Conference on Learning Representations*, volume 3, 2019.

Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi. Foundations for restraining bolts: Reinforcement learning with ltlf/ldlf restraining specifications. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 29, pp. 128–136, 2019.

Thomas Elsken, Jan Hendrik Metzen, Frank Hutter, et al. Neural architecture search: A survey. *J. Mach. Learn. Res.*, 20(55):1–21, 2019.

Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. 80:1406–1415, 2018.

Daniel Furelos-Blanco, Mark Law, Anders Jonsson, Krysia Broda, and Alessandra Russo. Induction and exploitation of subgoal automata for reinforcement learning. *Journal of Artificial Intelligence Research*, 70:1031–1116, 2021.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Felix Hill, Andrew Lampinen, Rosalia Schneider, Stephen Clark, Matthew Botvinick, James L McClelland, and Adam Santoro. Environmental drivers of systematicity and generalization in a situated agent. In *International Conference on Learning Representations*, 2020.

Felix Hill, Olivier Tieleman, Tamara von Glehn, Nathaniel Wong, Hamza Merzic, and Stephen Clark. Grounded language learning fast and slow. In *International Conference on Learning Representations*, 2021.

Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press, 2004.

Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith. Using reward machines for high-level task specification and decomposition in reinforcement learning. In *International Conference on Machine Learning*, pp. 2112–2121, 2018.

Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith. Learning reward machines for partially observable reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 15497–15508, 2019.

Kishor Jothimurugan, Suguman Bansal, Osbert Bastani, and Rajeev Alur. Compositional reinforcement learning from logical specifications. *Advances in Neural Information Processing Systems*, 34, 2021.

Yen-Ling Kuo, Boris Katz, and Andrei Barbu. Encoding formulas as deep networks: Reinforcement learning for zero-shot execution of ltl formulas. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5604–5610. IEEE, 2020.

Brenden Lake and Marco Baroni. Generalization without systematicity: On the compositional skills of sequence-to-sequence recurrent networks. In *International Conference on Machine Learning*, pp. 2873–2882, 2018.

Brenden M Lake. Compositional generalization through meta sequence-to-sequence learning. In *Advances in Neural Information Processing Systems*, pp. 9788–9798, 2019.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

Borja G. León and Francesco Belardinelli. Extended markov games to learn multiple tasks in multi-agent reinforcement learning. In *ECAI 2020 - 24th European Conference on Artificial Intelligence*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pp. 139–146. IOS Press, 2020. doi: 10.3233/FAIA200086. URL `https://doi.org/10.3233/FAIA200086`.

Michael L Littman, Ufuk Topcu, Jie Fu, Charles Isbell, Min Wen, and James MacGlashan. Environment-independent task specifications via gltl. *arXiv preprint arXiv:1704.04341*, 2017.

Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob N. Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language. In Sarit Kraus (ed.), *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pp. 6309–6317. ijcai.org, 2019. doi: 10.24963/ijcai.2019/880. URL `https://doi.org/10.24963/ijcai.2019/880`.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B. Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

James L McClelland, Matthew M Botvinick, David C Noelle, David C Plaut, Timothy T Rogers, Mark S Seidenberg, and Linda B Smith. Letting structure emerge: connectionist and dynamical systems approaches to cognition. *Trends in cognitive sciences*, 14(8):348–356, 2010.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pp. 1928–1937, 2016.

Junhyuk Oh, Satinder Singh, Honglak Lee, and Pushmeet Kohli. Zero-shot task generalization with multi-task deep reinforcement learning. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2661–2670. JMLR. org, 2017.

Paul Smolensky. On the proper treatment of connectionism. *Behavioral and brain sciences*, 11(1): 1–23, 1988.

Xingyou Song, Yuxiang Yang, Krzysztof Choromanski, Ken Caluwaerts, Wenbo Gao, Chelsea Finn, and Jie Tan. Rapidly adaptable legged robots via evolutionary meta-learning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2020, Las Vegas, NV, USA, October 24, 2020 - January 24, 2021*, pp. 3769–3776. IEEE, 2020. doi: 10.1109/IROS45743.2020.9341571. URL `https://doi.org/10.1109/IROS45743.2020.9341571`.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Open Ended Learning Team, Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021.

Rodrigo Toro Icarte, Toryn Q Klassen, Richard Valenzano, and Sheila A McIlraith. Teaching multiple tasks to an rl agent using ltl. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pp. 452–461. International Foundation for Autonomous Agents and Multiagent Systems, 2018.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeffrey Clune, and Kenneth Stanley. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In *International Conference on Machine Learning*, pp. 9940–9951. PMLR, 2020.

Haonan Yu, Haichao Zhang, and Wei Xu. Interactive grounded language acquisition and generalization in a 2d world. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net, 2018.

Lim Zun Yuan, Mohammadhosein Hasanbeig, Alessandro Abate, and Daniel Kroening. Modular deep reinforcement learning with temporal logic specifications. *arXiv preprint arXiv:1909.11591*, 2019.

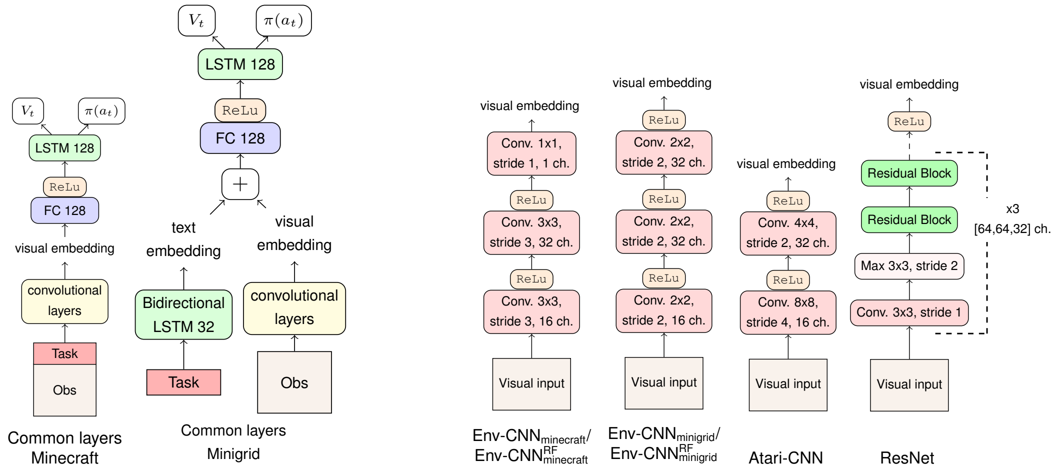
V_t
$\pi(a_t)$
LSTM 128
ReLu
FC 128
visual embedding
convolutional layers
Task
Obs
Common layers Minecraft
text embedding
visual embedding
Bidirectional LSTM 32
Common layers Minigrid
Conv. 1x1, stride 1, 1 ch.
Conv. 3x3, stride 3, 32 ch.
Conv. 3x3, stride 3, 16 ch.
Visual input
Env-CNN$_{minecraft}$/ Env-CNN$^{RF}_{minecraft}$
Conv. 2x2, stride 2, 32 ch.
Conv. 2x2, stride 2, 16 ch.
Env-CNN$_{minigrid}$/ Env-CNN$^{RF}_{minigrid}$
Conv. 4x4, stride 2, 32 ch.
Conv. 8x8, stride 4, 16 ch.
Atari-CNN
Residual Block
Max 3x3, stride 2
Conv. 3x3, stride 1
x3 [64,64,32] ch.
ResNet

Figure 2: The neural network architectures evaluated in Sec. 5. All the variants follow the structures depicted in the left for the corresponding settings. $\pi(a_t)$ and V_t refer to the actor and critic layers respectively Mnih et al. (2016). The different number of channels in the last layer of the Env-CNN architectures is done to provide evidence that such feature does not affect how the agent generalises.

A EXPERIMENT DETAILS

Here we provide further details about the experimental setting and training procedure.

A.1 ARCHITECTURE CONFIGURATIONS

Figure 2 includes schemes of the neural network architectures from Section 5. On the left, we see the general setting for Minecraft, where tasks instructions are embedded within the observation, and MiniGrid, where instructions are given in a separated channel. Note that we intentionally use a significantly different number of channels in the last layers with Env-CNN$_{\text{minecraft}}$ and Env-CNN$_{\text{minigrid}}$. We do this to evidence that the number of channels in the visual embedding does not affect the ability of SA architectures to generalise training instructions. In Sec 5 we observed that both SA architectures show good generalisation performance in their respective settings.

A.2 ENVIRONMENT

Map generation Training maps are procedurally-generated with a random number of objects that ranges from 2 to 8, there are no restrictions on the number of objects on a training map that can be the same as long as one of the objects solves the current task. As explained in the main text, the action set consists of four individual actions in Minecraft: *Up, Down, Left, Right*, that move the agent one slot from its current position according to the given direction; and three actions in MiniGrid: *Forward, Turn Left, Turn Right*. If the agent moves against a wall, its position remains the same. Note that moving against walls is discouraged by the internal reward function that provides a negative reward at each timestep if the agent has not solved the current task.

Instruction generation. As detailed in the main text, instructions are procedurally generated with objects from the corresponding set. For instance, with the small setting we have 6 different objects, meaning that agents are learning the negation operator from 6 training instructions. Consequently, the total number of training instructions in the small setting is 42 (6 positive, 6 negative and 30 disjunctions –we do not instruct disjunctions with the same object twice–).

Object-set generation. In the Minecraft-inspired environment, we procedurally generate objects as a random matrix of 9x9 values. Small and large training sets use 6 to 20 procedural objects as described in the main text. For validation and test sets we use 6 objects respectively. In MiniGrid, objects are not procedural values but compositions of colors, e.g., red, and shapes, e.g., key. Here the small set uses 2 shapes and 3 colors while training with negative instructions and disjunction. In validation we use an additional shape combined with training colors. In test we use 2 new colors, the validation shape and a new test shape. In the large setting, training has 4 shapes and 5 colors, an additional shape with training colors for validation, and 2 new colors with a a test shape and the validation shape in test.

Rewards. In both settings the agents received a penalisation of -1 when reaching an object that does not satisfy the task, and +1 when the opposite occurs. We explored different penalisation values per time-step to stop the agents from local-optima policies, e.g., not reaching any object, while still making beneficial avoiding undesired objects when possible. We did a grid-search in the range $[-0.01, -0.2]$ and achieved the best results with -0.1 in Minecraft and -0.05 in MiniGrid.

A.3 Training hyperparameters and plots

The A2C algorithm uses a learning rate of $8e^{-5}$, this value for the learning rate is the one that give us the best results when doing grid search between $1e^{-5}$ and $1e^{-3}$, except for the ResNet that shows better results with the value of $2e^{-5}$ that we use in the experiments. The entropy and value loss regularization terms are $1e^{-3}$ (best results we found in the range [$1e^{-4}$,0] and 0.5 (as in Mnih et al. (2016)) respectively. The expected-reward discount factor is 0.99, which is typical value for this term. Batch size is set to 512 values, which gives the best results in the range of ($[128, 2048]$).

Figure 3 shows the training and validation plots of the runs evaluated in Sec. 5. For each run we selected the weights that achieve the peak performance in the validation set.

A.4 Hardware

The various architectures were trained on computing clusters with GPUs such as Nvidia Tesla K80, Testla T4 or TITAN Xp. We employed Intel Xeon CPUs and consumed 7 GB of RAM per 5 independent runs working in parallel. Each experiment typically takes 2 days to train on this hardware. All the results presented in the main text and supplementary material are obtained from the aforementioned five independent runs per experiment.

B Generalising TTL formulae

In this section we include the pseudo-code of the of the symbolic module, whose procedures are explained in Sec. 4.1 of the main text and provide experimental results when agents generalise complex, i.e., multi-task, instructions.

Algorithm 1 details the Symbolic Module (SM), which given a TTL instruction ϕ, decomposes the instruction into atomic tasks T to be solved by the Neural Module (NM). Algorithm 2 details the extractor $\mathcal{E}$, an internal function of the SM that given the instruction formula generates a set $\mathcal{K}$ of sequences of tasks that satisfy ϕ. Algorithm 3 refers to the progression function $\mathcal{P}$, which updates $\mathcal{K}$ according to the true evaluation p given by the internal labelling function $\mathcal{L}_I$ that indicates to the SM how the previous task was solved.

B.1 Generalising complex TTL formulae

We use five complex instructions test the agents (see Table 3). The first two are the hardest specifications from Andreas et al. (2017); Toro Icarte et al. (2018). The three additional instructions were defined by aiming for combinations of negated tasks and inclusive/exclusive non-deterministic choices with non-atomic components together with sequential instances. We recall that operator $\cap$ can be defined in terms of sequential composition ; and non-deterministic choice $\cup$.

Generalisation with TTL formulae. We evaluate the performance of the same agents trained with atomic tasks T when executing complex (multi-task) instructions given as TTL formulae ϕ that refer to zero-shot atomic tasks. Table 4 shows the average reward and number of actions needed in 200 maps executing the complex instructions. In line with the results with TTL tasks from Table 1, we see that a stronger alignment yields better the final performance (i.e., SA>WA>NA accumulated reward).

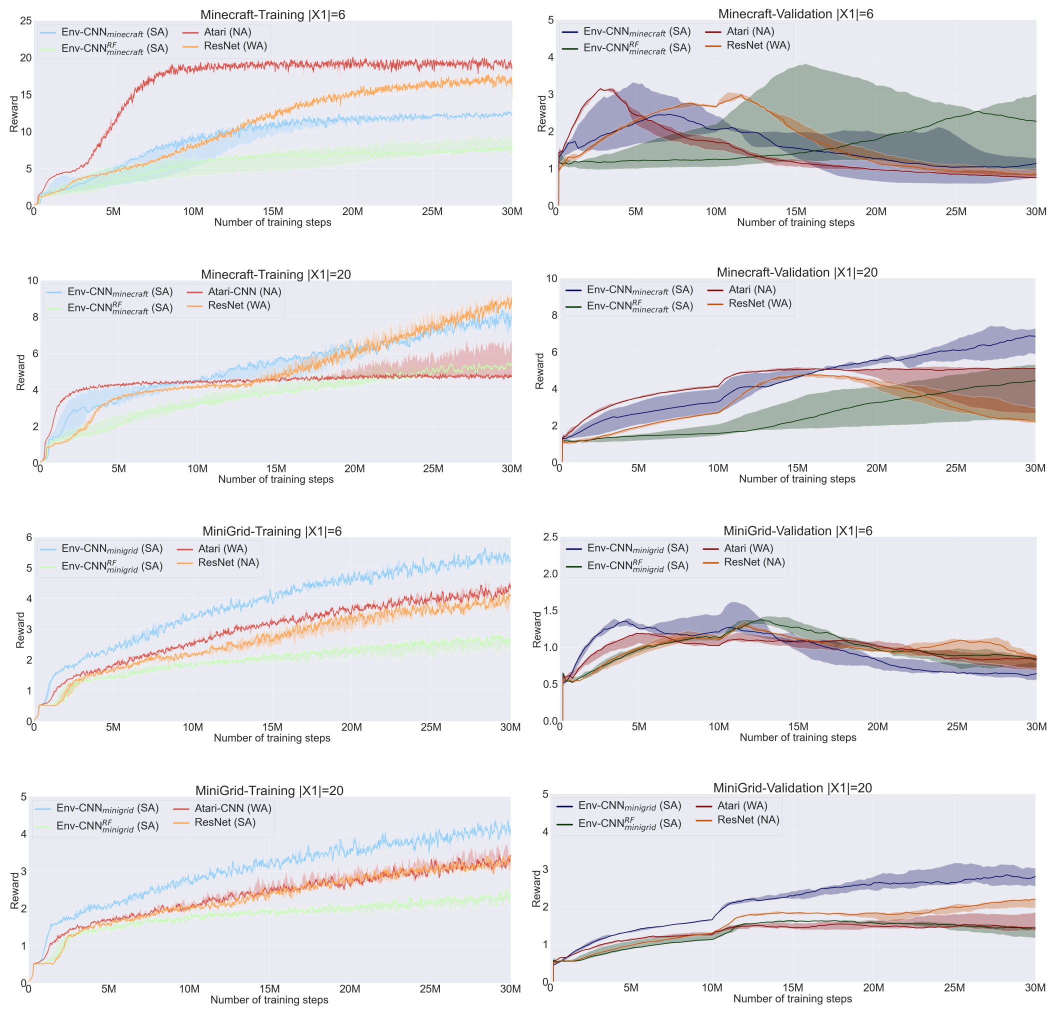

Figure 3: Train and validation plots with the different population sizes and 5 independent runs. Light colors refer to performance with training objects, whereas dark refer to validation objects. Lines refer to the median while shaded areas represent the 25-75 quartile difference.

Algorithm 1 The symbolic module (SM)

```
1: Input: Instruction $\phi$
2: Generate the accepted sequences of tasks $\mathcal{K} \leftarrow \mathcal{E}(\phi)$
3: Retrieve the current observation: $z$
4: Get the true proposition: $p \leftarrow \mathcal{L}_I(z)$
5: Get the first task: $T \leftarrow \mathcal{P}(\mathcal{K}, p)$
6: repeat
7:     Generate the extended observation: $z^{ext} \leftarrow z + T$
8:     Get the next action: $a \leftarrow \text{NM}(z^{ext})$
9:     Execute $a$, new observation $z'$
10:    New true proposition: $p' \leftarrow \mathcal{L}_I(z')$
11:    Provide the internal reward: NM $\leftarrow R_I(p')$
12:    if $p' == p_\alpha$ for $\alpha \in T$ then
13:        Update $\mathcal{K}, T' \leftarrow \mathcal{P}(\mathcal{K}, p')$
14:    end if
15: until $\mathcal{K} == \emptyset$ or time limit
```

Algorithm 2 Extractor function that obtains the list-of-lists with the possible sequences of tasks.

```
1: Function $\mathcal{E}$, Input: $\phi$
2: Initialize the list $\mathcal{K}$ with an empty sequences of tasks $Seq_0$
3: for each atomic task $T \in \phi$ do
4:     if $T$ is not a non-deterministic choice then
5:         for all $Seq \in \mathcal{K}$ do
6:             $Seq$.append($T$)
7:         end for
8:     else
9:         Save the first element: $FC \leftarrow T[0]$
10:        Save the last element: $SC \leftarrow T[-1]$
11:        Nseqs $\leftarrow length(\mathcal{K})$
12:        for all $Seq \in \mathcal{K}$ do
13:            Generate a clone: $Seq' \leftarrow Seq$
14:            $\mathcal{K}$.append($Seq'$)
15:        end for
16:        for $i$ in range($length(\mathcal{K})$) do
17:            if $i <$ Nseqs then
18:                $\mathcal{K}[i]$.append($FC$)
19:            else
20:                $\mathcal{K}[i]$.append($SC$)
21:            end if
22:        end for
23:    end if
24: end for
25: Output: $\mathcal{K}$
```

C Disjunction results

This section includes the results with disjunction anticipated in Sec. 5.

Table 5 shows the results of the different architectures when following reliable, i.e., pointing to the right object, and deceptive, i.e., pointing to the wrong object, zero-shot instructions with a non-deterministic choice. Instructions labelled “Disjunction 1st choice" refer to those where only the first object within the instruction is present in the map. “Disjunction 2nd choice" is a similar setting with the second object. In “Disjunction 2 choices" both objects are present in the map. As detailed in the main text, agents learning compositionally should be able to follow the zero-shot formulae and consequently do well with reliable instructions while poorly with deceptive ones.

Algorithm 3 Progression function that returns the next task to be solved.

```
1: Function $\mathcal{P}$, Input: $\mathcal{K}, p$
2: if $p \neq \emptyset$ then
3:     for all $Seq \in \mathcal{K}$ do
4:         if $p$ fulfills $Seq[0]$ then
5:             $Seq$.pop($Seq[0]$)
6:         else
7:             $\mathcal{K}$.pop($Seq$)
8:         end if
9:     end for
10: end if
11: $T \leftarrow \emptyset$
12: if $\mathcal{K}$ is not $\emptyset$ then
13:     Select the head $T$ of a sequence in $\mathcal{K}$:
14:     if $T \equiv \alpha$ then
15:         //Aiming to form a non-deterministic choice
16:         for all $Seq \in \mathcal{K}$ do
17:             if $T == Seq[0]$ then
18:                 continue
19:             else if $Seq[0] \equiv \alpha$ then
20:                 $T \leftarrow T \vee Seq[0]$
21:                 return $T$
22:             end if
23:         end for
24:     end if
25: end if
26: Output: $T$
```

Table 3: Complex instructions used in Table 4 in the main text. The 6 objects included in these instructions belong to the test set ($\mathcal{X}_3$) in the Minecraft-inspired environment.

TTL	Intuitive meaning
$((iron; workbench) \cap wood); toolshed; axe$	Get iron and then use workbench, also get wood. Then use the toolshed. Later use the axe.
$(wood \cap iron); workbench$	Get wood and iron. Then use the workbench
$(grass \sim\ ); grass; (workbench \cup toolshed)$	Get an object different from grass. Then get grass. Later use either the workbench or the toolshed.
$((workbench \sim\ ) \cap (toolshed \sim\ )); toolshed$	Use an object different from the workbench and use another object different from the toolshed. Then use the toolshed
$((wood; grass) \cup (iron; axe)); workbench; (toolshed \sim\ )$	Get either wood and then grass or iron and an axe. Then use the workbench. Later use something different from the toolshed.

D THE STRUGGLE WITH NEGATED INSTRUCTIONS

This section section aims to bring light to an apparent contradiction between our empirical results and those from the state-of-the-art about generalisation with negation Hill et al. (2020).

In Sec. 5 in the main text, we observe that the architecture of the CNN plays a key role in the ability of agents to generalise when the number of instructions is limited. Still, in Table 2 we see that the ResNet shows good generalisation in both settings when learning from 20 negated instructions. This seems to contradict the results from Hill et al. (2020) where the same ResNet struggles to generalise with 40 negated instructions. Nevertheless, we find that this is caused by the particular form that neural networks overfit to the negation operator.

Table 4: Results from a set of 200 maps with zero-shot complex instructions, i.e, longer than training instructions composed by various zero-shot tasks. Steps refers to the number of actions per instruction needed by the agents (a random walker requires 110 steps in average). The highest value over all networks is bolded.

	Atari-CNN (**NA**)		ResNet (**WA**)		Env-CNN$_{\text{minecraft}}$ (**SA**)		Env-CNN$^{\text{RF}}_{\text{minecraft}}$ (**SA**)	
$\lvert\mathcal{X}_1\rvert$	Reward	Steps	Reward	Steps	Reward	Steps	Reward	Steps
6	2.5 (0.3)	**26.3**(5.0)	2.7(0.7)	49.3(28.9)	**3.08**(1.5)	49.1(25.8)	2.7(1.8)	56.5(22.9)
20	3.0(0.2)	**21.0**(1.5)	3.5(0.7)	27.4(5.20)	**7.5**(3.4)	32.8(6.8)	5.5(1.2)	38.2(9.7)

Table 5: Complete results in BCMs with disjunctions. Results are bolded where performance with reliable instructions is greater or equal than two times the respective performance with deceptive instructions.

Minecraft	$\lvert\mathcal{X}_1\rvert$	Atari-CNN (**NA**)		ResNet (**WA**)		Env-CNN$_{\text{minecraft}}$ (**SA**)		Env-CNN$^{\text{RF}}_{\text{minecraft}}$ (**SA**)	
Task		Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive
Disj.	6	3.4 (0.1)	2.1(0.0)	2.9(0.3)	2.0(0.1)	**3.0**(1.4)	**0.8**(0.6)	**2.6**(1.8)	**1.2**(0.5)
1st choice	20	3.6(0.0)	2.2(0.0)	2.7(1.2)	1.8(1.0)	**5.1**(1.2)	**0.5**(0.5)	2.9(1.6)	1.0(0.8)
Disj.	6	3.3 (0.2)	2.1(0.0)	2.8(0.5)	2.0(0.2)	**3.1**(1.4)	**0.6**(0.5)	**2.6**(1.7)	**1.1**(0.5)
2nd choice	20	3.6(0.1)	2.2(0.0)	2.9(1.3)	1.7(0.2)	**4.9**(1.5)	**0.5**(0.6)	2.9(1.9)	1.0(0.7)
Disj.	6	2.6 (0.3)	2.6(0.6)	2.3(0.12)	2.5(0.7)	**2.2**(4.8)	**0.6**(0.3)	2.5(0.7)	1.4(0.1)
2 choices	20	2.6 (0.2)	2.7(0.6)	2.2(0.12)	2.2(0.7)	**3.5**(4.8)	**0.4**(0.3)	**2.8**(0.7)	**1.1**(0.9)

Minigrid	$\lvert\mathcal{X}_1\rvert$	Atari-CNN (**WA**)		ResNet (**NA**)		Env-CNN$_{\text{minigrid}}$ (**SA**)		Env-CNN$^{\text{RF}}_{\text{minigrid}}$ (**SA**)	
Task		Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive	Reliable	Deceptive
Disj.	6	**2.5** (0.6)	**0.9**(0.3)	3.7(0.6)	3.6(1.3)	**2.1**(1.2)	**0.9**(0.6)	**0.4**(0.9)	**0.8**(0.5)
1st choice	20	**3.0**(0.9)	**1.0**(0.4)	**6.1**(0.8)	**1.5**(0.8)	**2.6**(0.6)	**0.4**(0.1)	2.1(0.8)	1.3(0.3)
Disj.	6	**2.8** (0.9)	**0.9**(0.4)	3.5(0.8)	3.9(1.3)	**1.7**(1.3)	**0.8**(0.3)	**2.8**(0.8)	**0.9**(0.6)
2nd choice	20	2.1(1.4)	1.1(0.3)	2.6(0.7)	2.4(1.0)	**1.8**(0.5)	**0.5**(0.1)	1.9(0.8)	1.3(0.4)
Disj.	6	**4.2** (0.5)	**0.8** (0.5)	4.0(0.4)	3.7(0.6)	**2.7** (1.5)	**0.8**(0.5)	**4.1**(0.7)	**0.6**(0.4)
2 choices	20	**3.4**(1.0)	**1.1**(0.5)	**5.0**(0.6)	**1.9**(0.8)	**3.1**(0.5)	**0.4**(0.1)	2.7(1.0)	1.6(0.4)

In particular, consider the TTL task *copper* $\sim$, which intuitively means "get something different from copper". When agents overfit they no longer interpret the negation operator as a separated element, but rather as if the operator was part of the object's name. Consequently, for an overfitting agent, *copper* $\sim$ is a single word that represents all the objects of the training set except copper. This overfitting phenomena produces policies that, when facing non-training objects, are no longer able to interpret negation correctly, leading to a behavior different from what an agent learning systematically is expected to do. We find that this phenomena, first described in Hill et al. (2020), happens only *after* agents overfit to the training set. This means that through a validation set, we can perform early stopping – as we do in Sec. 5 – and use policies that correctly generalise to new instructions without requiring a larger number of training instructions.

Table 6 presents the results of an ablation study about the use of a validation set with a ResNet agent trained until convergence in the Minecraft large training set. We see that there is little difference with positive instructions, but results are completely opposite in the case of negated tasks, that go from an agent correctly generalising negation – when applying a validation set and early stopping – to one that does the opposite from what the specification requires – when no validation set is used –. Consequently, *the struggle with negated instructions in previous research seems to be motivated by the different ways deep learning overfits when learning different abstract operators*.

E Requirements for Logical Operators

Below we include additional details about the requirements we faced when aiming a neuro-symbolic agent that generalises the negation and non-deterministic choice operators.

Table 6: Results from a ResNet in the Minecraft setting when using a validation set or when not, i.e., the latter are the results from an agent where we did not perform early stopping and overfitted. We see that the different behaviours with positive and negative when the agent is overfitting align with the results from Hill et al. (2020). Results are bolded where performance with reliable instructions is greater or equal than two times the respective performance with deceptive instructions.

	No Validation Set		Validation Set	
ResNet ($|\mathcal{X}_1| = 20$)	Reliable	Deceptive	Reliable	Deceptive
Positive instructions	**3.1**(1.0)	**1.3**(0.7)	**3.3**(0.5)	**1.1**(0.8)
Negated instructions	1.2(0.5)	1.6(0.7)	**3.1**(0.5)	**1.3**(0.9)

E.1 Negation

When learning positive and negative instances concurrently using visual instructions, we noticed that the position of the negation operator impacted on the agent's behavior in training. Specifically, placing the negation symbol after the negated object helps the agent to learn both types of instances concurrently, while, when placed otherwise, the agent focus only on the positive tasks first, which can lead to saddle points.

We believe that this result is not related to linguistic processing. Note that all of our architectures in Minecraft process the specification as a whole, as part of the observational input processed by the convolutional layer. Thus, the order is not important from the point of view of temporal processing. We believe the difference was caused because affirmative formulas are represented with one symbol, e.g., "get wood" is represented as "wood"; while negation is represented by two symbols, e.g., "get an object different from wood" as "wood $\sim$". By placing the negation operator after the object we force the neural network to "observe" the first two inputs to differentiate affirmative tasks from negative ones. An alternative approach that could have a similar effect is adding an explicitly positive operator so that both negative and positive formulas have the same length.

E.2 Non-deterministic Choice

Training the agent to effectively learn non-deterministic choice $\cup$ required episodes where only one of the two disjuncts of the non-deterministic choice was present, as well as some other episodes with the two objects present simultaneously. Intuitively, if we have the instruction "get axe or get gold", there should be training episodes where gold is present in the map, but there are no axes, and similarly, other maps where there are axes, but there is not gold. This prevents the agent from biasing with the object that appears either in the first or second position of the instruction. Additionally, the agent should also interact with episodes where both objects are present. Otherwise, the policy network of the agent will allocate a similarly high probability to go after the two objects, instead of to the one that is closer or "safer", i.e, far from non-desired objects.

F Translating TTL into LTL$_f$

It can be easily inferred that our task temporal logic (TTL) is a fragment of the more popular linear-time temporal logic over finite traces (LTL$_f$) Baier & Katoen (2008). The syntax of LTL$_f$ is defined as follows:

$$\varphi \ ::= \ p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \bigcirc\varphi \mid \varphi_1 \mathrm{U} \varphi_2$$

Note that both TTL and LTL$_f$ are defined over finite traces. Below we provide truth-preserving translations of TTL into LTL$_f$.

Definition 4. *Translation τ from TTL to LTL$_f$ is defined as follows:*

$$\begin{aligned}
\tau(\alpha) &= \diamond p_\alpha, \textit{ where } p_\alpha \textit{ is the fulfilment condition associated with task } \alpha \\
\tau(\alpha\sim) &= \diamond(\textstyle\bigvee_{\beta\neq\alpha} p_\beta \wedge \neg p_\alpha) \\
\tau(\alpha \vee \alpha') &= \tau(\alpha) \vee \tau(\alpha')
\end{aligned}$$

$$
\begin{array}{rcl}
\tau(T \cup T') & = & \tau(T) \vee \tau(T') \\
\tau(T; T') & = & \begin{cases} \diamond(p_\alpha \wedge \tau(T)) & \textit{if } T = \alpha \\ \diamond((\bigvee_{\beta \neq \alpha} p_\beta \wedge \neg p_\alpha) \wedge \tau(T)) & \textit{if } T = \alpha\sim \\ \diamond((p_\alpha \vee p_{\alpha'}) \wedge \tau(T)) & \textit{if } T = \alpha \vee \alpha' \\ \tau(T_1; (T_2; T')) & \textit{if } T = T_1; T_2 \\ \tau((T_1; T') \cup (T_2; T')) & \textit{if } T = T_1 \cup T_2 \end{cases}
\end{array}
$$

We immediately prove that translation τ preserve the interpretation of formulae in TTL.

Proposition 1. *Given a model $\mathcal{N}$ and trace λ, for every formula T in TTL,*

$$(\mathcal{N}, \lambda) \models T \quad \textit{iff} \quad (\mathcal{N}, \lambda) \models \tau(T)$$

In the main body of this work we stated that TTL is a fragment of LTL$_f$, and supported such claim by explaining that translations τ_1 and τ_2 preserve the interpretation of formulae in TTL. We include now a detailed proof for this claim:

Proposition 2. *Given a model $\mathcal{N}$ and trace λ, for every formula T in TTL,*

$$(\mathcal{N}, \lambda) \models T \quad \textit{iff} \quad (\mathcal{N}, \lambda) \models \tau(T)$$

Proof. The proof is by induction on the structure of formula T. The base case follows immediately by the semantics of TTL and LTL$_f$.

As for the induction step, the case of interest is for formulae of type $T; T'$. In particular, consider the case for $T = \alpha$. We then have that $(\mathcal{N}, \lambda) \models \alpha; T'$ iff for some $0 \leq j < |\lambda|$, $(\mathcal{N}, \lambda[0, j]) \models \alpha$ and $(\mathcal{N}, \lambda[j+1, |\lambda|]) \models T'$. By induction hypothesis, this is equivalent to $(\mathcal{N}, \lambda[0, j]) \models \diamond p_\alpha$ and $(\mathcal{N}, \lambda[j+1, |\lambda|]) \models \tau(T')$. Finally, this is equivalent to $(\mathcal{N}, \lambda) \models \diamond(p_\alpha \wedge \tau(T'))$. The other cases. are dealt with similarly.

Finally, the case for $T \cup T'$ follows by induction hypothesis and the distributivity of $\diamond$ over $\vee$. $\square$