Improving Intrinsic Exploration with Language Abstractions

Jesse Mu[1]*, **Victor Zhong**[2,3], **Roberta Raileanu**[3], **Minqi Jiang**[3,4], **Noah Goodman**[1], **Tim Rocktäschel**[4*], **Edward Grefenstette**[4,5*]
[1]Stanford University, [2]University of Washington, [3]Meta AI, [4]University College London, [5]Cohere

Abstract

Reinforcement learning (RL) agents are particularly hard to train when rewards are sparse. One common solution is to use *intrinsic* rewards to encourage agents to explore their environment. However, recent intrinsic exploration methods often use state-based novelty measures which reward low-level exploration and may not scale to domains requiring more abstract skills. Instead, we explore *language* as a general medium for highlighting relevant abstractions in an environment. Unlike previous work, we evaluate whether language can improve over existing exploration methods by directly extending (and comparing to) competitive intrinsic exploration baselines: AMIGo (Campero et al., 2021) and NovelD (Zhang et al., 2021). These language-based variants outperform their non-linguistic forms by 47–85% across 13 challenging tasks from the MiniGrid and MiniHack environment suites.

1 Introduction

A central challenge in reinforcement learning (RL) is designing agents that can solve complex, long-horizon tasks with sparse rewards. In the absence of extrinsic rewards, one popular solution is to provide *intrinsic* rewards for exploration [34, 35, 42, 43]. This invariably leads to the challenging question: how should one measure exploration? One common answer is that an agent should be rewarded for attaining "novel" states in the environment, but naive measures of novelty have limitations. For example, consider an agent that starts in the kitchen of a large house and must make an omelet. Simple state-based exploration will reward an agent for visiting every room in the house, but a more effective strategy would be to stay put and use the stove. Moreover, like kitchens with different-colored appliances, states can look cosmetically different but have the same underlying semantics, and thus are not truly novel. Together, these constitute two fundamental challenges for intrinsic exploration: first, how can we reward true progress in the environment over meaningless exploration? Second, how can we tell when a state is not just superficially, but *semantically* novel?

Fortunately, humans are equipped with a powerful tool for solving both problems: language. As a cornerstone of human intelligence, language has strong priors over the features and behaviors needed for exploration and skill acquisition. It also describes a rich and compositional set of meaningful behaviors as simple as directions (e.g. *move left*) and as abstract as conjunctions of high level tasks (e.g. *retrieve the ring and defeat the wizard*) that can categorize and unify many possible world states.

Our aim is to see whether language abstractions can improve existing state-based exploration methods in RL. While language-guided exploration methods exist in the literature [3, 5, 12, 13, 21–24, 31, 44, 51, 53], we make two key contributions over prior work. First, existing methods assume access to a high-level linguistic instruction for reward shaping, or otherwise assume that any intermediate language annotations encountered are always helpful for learning. Instead, we study settings without

*Work done while at Meta AI. Correspondence to muj@stanford.edu

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

instructions, with more diverse intermediate messages (Figure 1) that may or may not be useful, but may nonetheless be a more effective measure of novelty than raw states.

Second, past work often compares only to vanilla RL, while ignoring competitive intrinsic exploration baselines. This leaves the true utility of language over simpler state-based exploration unclear. To remedy this issue, we conduct a controlled evaluation on the effect of language on competitive approaches to exploration by extending two recent, state-of-the-art methods: AMIGo [7], where a teacher proposes intermediate location-based goals for a student, and NovelD [54], which rewards an agent for visiting novel regions of the state space. Building upon these methods, we propose **L-AMIGo**, where the teacher proposes goals expressed via language instead of coordinates, and **L-NovelD**, a variant of NovelD with an additional exploration bonus for visiting linguistically-novel states. Across 13 challenging, procedurally-generated, sparse-reward tasks in the MiniGrid [8] and MiniHack [41] environment suites, we show that language-parameterized exploration methods outperform their non-linguistic counterparts by 47–85%, especially in more abstract tasks with larger state and action spaces. We also show that language improves the interpretability of the training process, either by developing a natural curriculum of semantic goals (in L-AMIGo) or by allowing us to visualize the most novel language during training (in L-NovelD). Finally, we show when and where the fine-grained compositional semantics of the language improves agent exploration, when compared to non-compositional baselines.

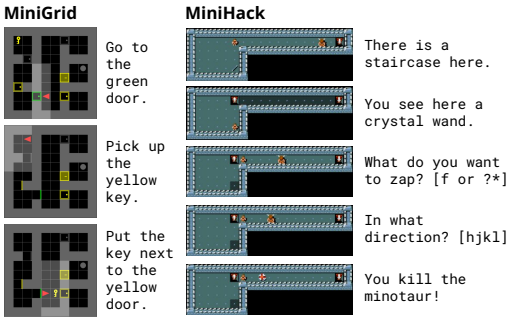


Figure 1: **Language conveys meaningful environment abstractions.** Language state annotations in the MiniGrid KeyCorridorS4R3 [8] and MiniHack Wand of Death (Hard) [41] tasks.

2 Related Work

Exploration in RL. Exploration has a long history in RL, from ε-greedy [48] or count-based exploration [4, 29, 30, 32, 47, 50] to intrinsic motivation [33–35] and curiosity-based learning [42]. More recently, deep neural networks have been used to measure novelty with changes in state representations [6, 40, 54] or prediction errors in world models [1, 36, 46]. Another long tradition generates curricula of intrinsic goals to encourage learning [7, 13, 14, 18–20, 37–39]. In this paper, we explore the potential benefit of language on these approaches to exploration.

Language for Exploration. The observation that language-guided exploration can improve RL is not new: language has been used to shape policies [24, 51] and rewards [3, 5, 21–23, 31, 44, 53] and set intrinsic goals [12, 13]. Crucially, our work differs from prior work in two ways: first, instead of reward shaping with high-level instructions, we use noisier, intermediate language annotations for exploration; second, we directly extend and compare to competitive intrinsic exploration baselines.

L-AMIGo, our variant of AMIGo with language goals, is similar to the IMAGINE agent of Colas et al. [13], which also sets intrinsic language goals. However, IMAGINE is built for instruction following, and requires a perfectly compositional space of language goals, which the agent tries to explore so that it can complete novel goals at test time. Instead, we make no assumptions on the language and explore to maximize extrinsic reward, using an alternative *goal difficulty* metric to measure progress.

Meanwhile, reward shaping and inverse RL methods [3, 5, 21–24, 31, 44, 51, 53] reward an agent for actions associated with linguistic descriptions, but again are primarily designed for instruction following, where an extrinsic goal is available to help shape intermediate rewards. In our setting, however, we have not high-level extrinsic goals but low-level *intermediate* language annotations. Extrinsic reward shaping methods such as LEARN [22] could be naively applied by simply doing reward shaping with every intermediate language annotation, and a few of these methods are designed for low-level language subgoals [24, 31]. However, a shared assumption of these approaches is that *language is always helpful*: either because we have expert-curated messages (as in Harrison et al. [24]), or because we have goal descriptions that let us identify subgoals relevant to the extrinsic goal (as in ELLA; Mirchandani et al. [31]). In our tasks, however, most language is *unhelpful* for progress in the environment, and we have no extrinsic goals. Consequently, past methods reduce to

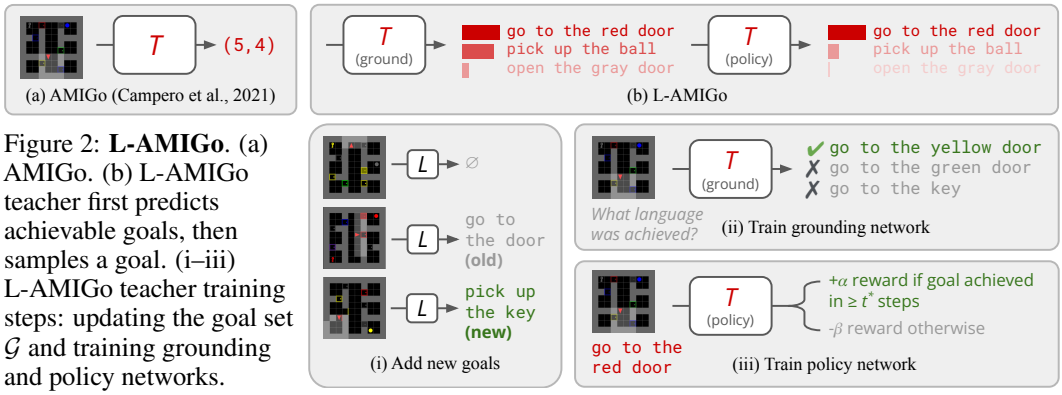


Figure 2: **L-AMIGo**. (a) AMIGo. (b) L-AMIGo teacher first predicts achievable goals, then samples a goal. (i–iii) L-AMIGo teacher training steps: updating the goal set $\mathcal{G}$ and training grounding and policy networks.

simply giving a fixed reward for every intermediate message encountered, which (we will show) fails to learn. Finally, work concurrent to ours by Tam et al. [49] tackles similar ideas in photorealistic environments that permit transfer from foundation models. Instead, we explore domain-specific symbolic games with built-in language where such models are not readily available.

A final distinguishing contribution of our work is that prior work often neglects non-linguistic exploration baselines. For example, Harrison et al. [24] and LEARN [22] compare to vanilla RL only; ELLA [31] compares to LEARN and RIDE [40], with limited improvements over RIDE. Prior work can thus be summarized as showing that linguistic rewards improve over extrinsic rewards alone. Instead, we provide novel evidence that *linguistic rewards improve upon state-based intrinsic rewards*, using the same exploration methods and challenging tasks typical of recent work in RL.

3 Problem Statement

We explore RL in the setting of an augmented Markov Decision Process (MDP) defined as the tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma, L)$, where $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, $T : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the environment transition dynamics, $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the *extrinsic* reward function, where $r_t = R(s_t, a_t)$ is the reward obtained at time t by taking action a_t in state s_t, and γ is the discount factor. To add language, we assume access to an *annotator* L that produces **language descriptions** for states: $\ell_t = L(s_t)$, such as those in Figure 1. Note that not every state needs a description (which we model with a null description $\varnothing$) and the set of descriptions need not be known ahead of time.[2] We ultimately seek a policy that maximizes the expected discounted (extrinsic) reward $R_t = \mathbb{E}[\sum_{k=0}^{H} \gamma^k r_{t+k}]$, where H is the finite time horizon. During training, however, we maximize an *augmented* reward $r_t^+ = r_t + \lambda r_t^i$, where r_t^i is an *intrinsic* reward and λ is a scaling hyperparameter.

Like past work [26, 31, 53] we make the simplifying assumption of access to an oracle language annotator L provided by the environment. Note that the language annotator is "oracle" in that it always outputs messages that are true of the current state, but *not* "oracle" in that it indiscriminately outputs messages that are not necessarily relevant for the extrinsic goal. Many modern RL environments are pre-equipped with language, including NetHack/MiniHack [28, 41], text-based games [15, 45, 52], and in fact most video games in general. In the absence of an oracle annotator, one common approach is to learn an annotator model from a dataset of language-annotated states [3, 22, 31], though such datasets are often generated from oracles that are simply run offline instead [31]. Concurrent work [49] uses pretrained foundation models to automatically provide annotations in 3D environments, though such models are not readily available in the symbolic 2D games we explore. Since this idea has been well-proven, we assume oracle access to L, but as an example, one could straightforwardly adapt the annotator model trained on BabyAI by Mirchandani et al. [31] to our setting.

[2]For presentational simplicity, the annotator here outputs a single description per state, but in practice, we allow an annotator to produce *multiple* descriptions: e.g. in MiniGrid, *open the door* and *open the red door* describe the same state. This requires two minor changes in the equations, described in Footnotes 4 and 5.

4 L-AMIGo

We now describe our approach to jointly training a student and a goal-proposing teacher, extending AMIGo [7] to arbitrary language goals.

4.1 Adversarially Motivated Intrinsic Goals (AMIGo)

AMIGo [7] augments an RL student policy with goals generated by a teacher, which provide intrinsic reward when completed (Figure 2a). The idea is that the teacher should propose intermediate goals that start simple, but grow harder to encourage an agent to explore its environment.

Student. Formally, the student S is a *goal-conditioned* policy parameterized as $\pi_S(a_t \mid s_t, g_t; \theta_S)$, where g_t is the goal provided by the teacher, and the student receives an intrinsic reward r_t^i of 1 only if the teacher's goal at that timestep is completed. The student receives a goal from the teacher either at the beginning of an episode, or mid-episode, if the previous goal has been completed.

Teacher. Separately, AMIGo trains an adversarial *teacher* policy $\pi_T(g_t \mid s_0; \theta_T)$ to propose goals to the student given its initial state. The teacher is trained with a reward r_t^T that depends on a *difficulty threshold* t^*: the teacher is given a positive reward of $+\alpha$ for proposing goals that take the student more than t^* timesteps to complete, and $-\beta$ for goals that are completed sooner, or never completed within the finite time horizon. To encourage proposing harder and harder goals that promote exploration, t^* is increased linearly throughout training: whenever the student completes 10 goals in a row under the current difficulty threshold, it is increased by 1, up to some tunable maximum difficulty. Finally, to encourage intermediate goals that are aligned with the extrinsic goal, the teacher is also rewarded with the extrinsic reward when the student attains it.

This teacher is updated separately from the student at different time intervals. Formally, its training data is batches of (s_0, g_t, r_t^T) tuples collected from student trajectories for nonzero r_t^T, where s_0 is the initial state of the student's trajectory and g_t is the goal that led to reward r_t^T.

The original paper [7] implements AMIGo for MiniGrid only, where the goals g_t are (x, y) coordinates to be reached. The student gets the goal embedded directly in the $M \times N$ environment, and the teacher is a dimensionality-preserving convolutional network which encodes the student's $M \times N$ environment into an $M \times N$ distribution over coordinates, from which a single goal is selected.

4.2 Extension to L-AMIGo

Student. The L-AMIGo student is a policy conditioned not on (x, y) goals, but on *language goals* ℓ_t: $\pi_S(a_t \mid s_t, \ell_t; \theta_S)$. Given the "goal" ℓ_t, the student is now rewarded if it reaches a state with the language description ℓ_t, i.e. if $\ell_t = L(s_t)$.[3]. Typically this student will encode the goal with a learned language model and concatenate the goal representation with its state representation.

Teacher. Now the L-AMIGo teacher selects goals from the set of possible language descriptions in the environment. Because the possible goals are initially unknown, the teacher maintains a running set of goals $\mathcal{G}$ that is updated as the student encounters new state descriptions (Figure 2i).

This move to language creates a challenge: not only must a teacher choose a goal to propose, it must also determine which goals are achievable at all. For example, the goal *go to the red door* only makes sense in environments with red doors. In L-AMIGo, these tasks are factorized into a **policy network**, which produces the distribution over goals given a student's state, and a **grounding network**, which predicts the probability that a goal is likely to be achieved in the first place (Figure 2b):

$$\pi_T(\ell_t \mid s_t; \theta_T) \propto p_{\text{ground}}(\ell_t \mid s_t; \theta_T) \cdot p_{\text{policy}}(\ell_t \mid s_t; \theta_T) \tag{1}$$

$$p_{\text{ground}}(\ell_t \mid s_t; \theta_T) = \sigma\left(f(\ell_t; \theta_T) \cdot h_{\text{ground}}(s_t; \theta_T)\right) \tag{2}$$

$$p_{\text{policy}}(\ell_t \mid s_t; \theta_T) \propto f(\ell_t; \theta_T) \cdot h_{\text{policy}}(s_t; \theta_T) \tag{3}$$

[3]We can treat language *goals* and *state descriptions* equivalently, even if the wordings are slightly different across environments. In MiniGrid, messages (e.g. *go to the red door*) look like goals but can also be interpreted as state descriptions: *[in this state, you have] go[ne] to the red door* In MiniHack, messages are description-like (e.g. *you kill the minotaur!*), but imagine the teacher's goal as *[reach a state where] you kill the minotaur!*

Equation 3 describes the policy network as producing a probability for a goal by computing the dot product between goal and state representations $f(\ell_t; \theta_T)$ and $h_{\text{policy}}(s_t; \theta_T)$, normalizing over possible goals; this policy is learned identically to the standard AMIGo teacher (Figure 2iii). Equation 2 specifies the grounding network as predicting whether a goal is *achievable* in an environment, by applying the sigmoid function to the dot product between the goal representation $f(\ell_t; \theta_T)$ and a (possibly separate) state representation $h_{\text{ground}}(s_t; \theta_T)$. Given an oracle grounding classifier, which outputs only 0 or 1, this is equivalent to restricting the teacher to proposing only goals that are achievable in a given environment. In practice, however, we learn the classifier online (Figure 2ii). Given the initial state s_0 of an episode, we ask the grounding network to predict the first language description encountered along this trajectory: $\ell_{\text{1st}} = L(s_{t'})$, where t' is the minimum t where $L(s_t) \neq \varnothing$. This is formalized as a multilabel binary cross entropy loss,

$$\mathcal{L}_{\text{ground}}(s_0, \ell_{\text{1st}}) = -\log(p_{\text{ground}}(\ell_{\text{1st}} \mid s_0; \theta_T)) - \frac{1}{|\mathcal{G}|-1} \sum_{\ell' \in \mathcal{G} \setminus \{\ell_{\text{1st}}\}} \log(1 - p_{\text{ground}}(\ell' \mid s_0; \theta_T)), \quad (4)$$

where the second term noisily generates negative samples of (start state, unachieved description) pairs based on the set of descriptions $\mathcal{G}$ known to the teacher at the time, similar to contrastive learning.[4] Note that since $\mathcal{G}$ is updated during training, Equation 4 grows to include more terms over time.

To summarize, training the teacher involves three steps: (1) updating the running set of descriptions seen in the environment, (2) learning the policy network based on whether the student achieved goals proposed by the teacher, and (3) learning the grounding network by predicting descriptions encountered from initial states. Algorithm S1 in Appendix A describes how L-AMIGo trains in an asynchronous actor-critic framework, where the student and teacher are jointly trained from batches of experience collected from separate actor threads, as used in our experiments (see Section 6).

5 L-NovelD

Next, we describe NovelD [54], which extends simpler tabular- [47] or pseudo- [4, 6] count-based intrinsic exploration methods, and our language variant, L-NovelD. Instead of simply rewarding an agent for rare states, NovelD rewards agents for *transitioning* from states with low novelty to states with higher novelty. Zhang et al. [54] show that NovelD surpasses Random Network Distillation [6], another popular exploration method, on a variety of tasks including MiniGrid and Atari.

5.1 NovelD

NovelD defines the reward r_t^i to be the difference in novelty between state s_t and previous state s_{t-1}:

$$r_t^i = \text{NovelD}_s(s_t, s_{t-1}) \triangleq \underbrace{\max(N(s_t) - \alpha N(s_{t-1}), 0)}_{\text{Term 1 (NovelD)}} \cdot \underbrace{\mathbb{1}(N_e(s_t) = 1)}_{\text{Term 2 (ERIR)}}. \quad (5)$$

In the first NovelD term, $N(s_t)$ is the novelty of state s_t; this quantity describes the difference in novelty between successive states, which is clipped > 0 so the agent is not penalized from moving back to less novel states. α is a hyperparameter that scales the average magnitude of the reward. The second term is the **E**pisodic **R**eduction on **I**ntrinsic **R**eward (ERIR): a constraint that the agent only receives reward when encountering a state *for the first time in an episode*. $N_e(s_t)$ is an episodic state counter that tracks exact state visitation counts, as defined by (x, y) coordinates.

Measuring novelty with RND. In smaller MDPs, it is possible to track exact state visitation counts, in which case the novelty is typically the inverse square root of visitation counts [47]. However, in larger environments where states are rarely revisited, we (like NovelD) use the popular Random Network Distillation (RND) [6] technique as an approximate novelty measure. Specifically, the novelty of a state is measured by the prediction error of a state embedding network that is trained jointly with the agent to match the output of a fixed, random target network. The intuition is that states which the RND network has been trained on will have lower prediction error than novel states.

[4] If multiple "first" descriptions are found, the teacher predicts 1 for *each* description, and 0 for all others.

5.2 Extension to L-NovelD

Our incorporation of language is simple: we add an additional exploration bonus based on novelty defined according to the language descriptions of states:

$$\text{NovelD}_\ell(\ell_t, \ell_{t-1}) \triangleq \max(N(\ell_t) - \alpha N(\ell_{t-1}), 0) \cdot \mathbb{1}(N_e(\ell_t) = 1). \tag{6}$$

This bonus is identical to standard NovelD: $N(\ell)$ is the novelty of the description ℓ as measured by a *separately parameterized* RND network encoding the description,[5] and $N_e(\ell_t) = 1$ when the language description has been encountered for the first time this episode. We keep the original NovelD exploration bonus, as language rewards may be sparse and a basic navigation bonus can encourage the agent to reach language-annotated states. The final intrinsic reward for L-NovelD is

$$r_t^i = \text{L-NovelD}(s_t, s_{t-1}, \ell_t, \ell_{t-1}) \triangleq \text{NovelD}_s(s_t, s_{t-1}) + \lambda_\ell \text{NovelD}_\ell(\ell_t, \ell_{t-1}) \tag{7}$$

where λ_ℓ controls the trade-off between Equations 5 and 6.

One might ask why we do not simply include the language description ℓ as input into the RND network, along with the state. While this can work in some cases, decoupling the state and language novelties allow us to precisely control the trade-off between the two, with a hyperparameter that can be tuned to different tasks. In contrast, a combined input obfuscates the relative contributions of state and language to the overall novelty. Appendix F.2 has ablations that show that (1) combining the state and language inputs or (2) using the language novelty term alone leads to worse performance.

6 Experiments

We evaluate L-AMIGo, AMIGo, L-NovelD, and NovelD, implemented in the TorchBeast [27] implementation of IMPALA [17], a common asynchronous actor-critic method. Besides vanilla IMPALA, we also compare to a naive (fixed) message reward given for any message in the environment, which is similar doing extrinsic reward shaping for all messages (e.g. LEARN [22]; also [3, 5, 21, 23, 44, 53]) or prior approaches that assume that messages are always helpful (Harrison et al. [24], ELLA [31]); see Appendix B for more discussion on this baseline and its equivalencies to prior work.[6] We run each model 5 times across 13 tasks within two challenging procedurally-generated RL environments, MiniGrid [8] and MiniHack [41], and adapt baseline models provided for both environments [7, 41]; for full model, training, and hyperparameter details, see Appendix C.

6.1 Environments

MiniGrid. Following Campero et al. [7], we evaluate on the most challenging tasks in MiniGrid [8], which involve navigation and manipulation tasks in gridworlds: **KeyCorridorS{3,4,5}R3** (Figure 1) and **ObstructedMaze_{1Dl,2Dlhb,1Q}**. These tasks involve picking up a ball in a locked room, with the key to the door hidden in boxes or other rooms and the door possibly obstructed. The suffix indicates the size of the environment, in increasing order. See Appendix G.1 for more details.

To add language, we use the complementary BabyAI platform [9] which provides a grammar of 652 possible messages, involving *goto*, *open*, *pickup*, and *putnext* commands applied to a variety of objects qualified by type (e.g. *box*, *door*) and/or color (e.g. *red*, *blue*). The oracle language annotator emits a message when the corresponding action is completed. On average, only 6 to 12 messages (1-2% of all 652) are needed to complete each task (see Appendix G.1 for all messages).

Note that since BabyAI messages are not included in the original environment from which we adapt baseline agents [7], none of our MiniGrid agents encode language observations directly into the state. While it can be tempting and beneficial to use language in this way, one *a priori* benefit of using language solely for exploration is that language is only needed during training, and not evaluation. Regardless, see Appendix E for additional experiments with MiniGrid agents that encode language into the state representation; while this boosts performance of baseline models, the experiments show that language-augmented exploration methods still outperform non-linguistic ones.

[5] For multiple messages, we average NovelD of each messsage.

[6] For an implementation of a message reward with simple novelty-based decay, see the message-only L-NovelD ablation results in Appendix F.2, which underperforms full L-NovelD and L-AMIGo.

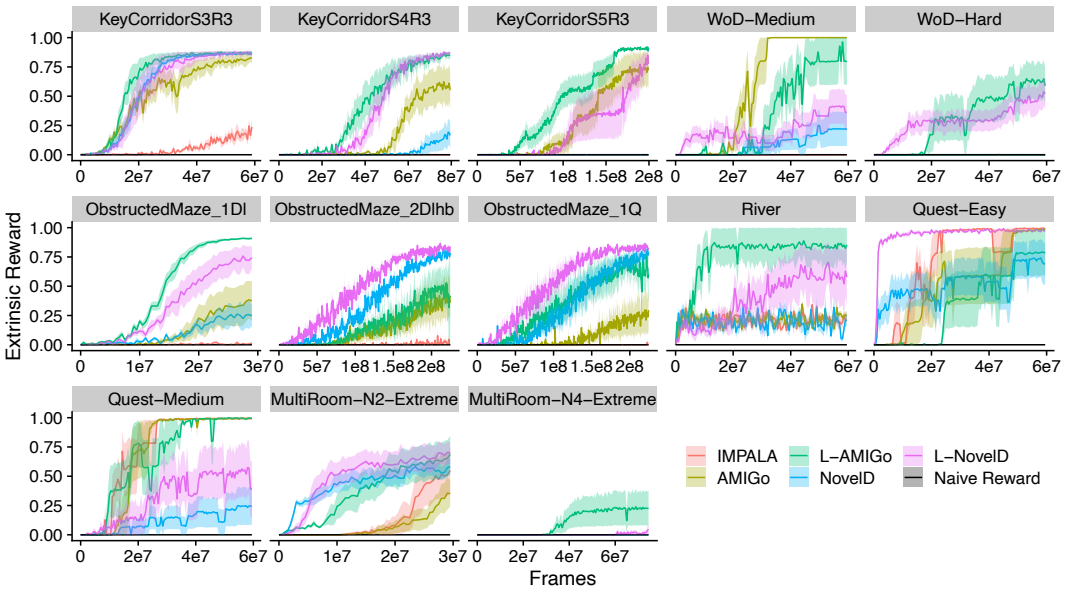


Figure 3: **Training curves**. Mean extrinsic reward ($\pm$ std err) across 5 independent runs for each model and environment. In general, linguistic variants outperform their non-linguistic forms.

MiniHack. MiniHack [41] is a suite of procedurally-generated tasks of varying difficulty set in the roguelike game NetHack [28]. MiniHack contains a diverse action space beyond simple MiniGrid-esque navigation, including planning, inventory management, tool use, and combat. These actions cannot be expressed by (x, y) positions, but instead are captured by in-game messages (Figure 1). We evaluate our methods on a representative suite of tasks of varying difficulty: **River**, **Wand of Death (WoD)-{Medium,Hard}**, **Quest-{Easy,Medium}**, and **MultiRoom-{N2,N4}-Extreme**.

For space reasons, we describe the WoD-Hard environment here, but defer full descriptions of tasks (and messages) to Appendices G.2 and H. In WoD-Hard, depicted in Figure 1, the agent must learn to use a *Wand of Death*, which can zap and kill enemies. This involves a complex sequence of actions: the agent must find the wand, pick it up, choose to *zap* an item, select the wand in the inventory, and finally choose the zapping direction (towards the minotaur which is pursuing the player). It must then proceed past the minotaur to the goal to receive reward. Taking these actions out of order (e.g. trying to *zap* with nothing in the inventory, or selecting something other than the wand) has no effect.

It is difficult to enumerate all MiniHack messages, as they are hidden in low-level game code which has many edge cases. As an estimate, we can examine expert policies: agents which have solved WoD tasks encounter around 60 messages, of which only 5–10 (8–16%) are needed for successful trajectories, including inventory (*f - a metal wand*, *what do you want to zap?*, *in what direction?*) and combat (*You kill the minotaur!*, *Welcome to level 2.*) messages; most are irrelevant (e.g. picking up and throwing stones) or nonsensical (*There is nothing to pick up*, *That is a silly thing to zap*). In the other tasks, only 8–18% of the hundreds of unique messages are needed for success (Appendix G.2).

Unlike the MiniGrid environments, we adapt baseline models from [41], which all already encode the in-game message into the state representation. Despite this, as we will show, using language as an explicit target for exploration outperforms using language as a state feature alone.

7 Results

Figure 3 shows training curves with AMIGo, NovelD, language variants, and the IMPALA and naive message reward baselines. Following Agarwal et al. [2], we summarize these results with the interquartile mean (IQM) of all methods in Figure 4, with bootstrapped 95% confidence intervals constructed from 5k samples per model/env combination.[7][8] We come to the following conclusions:

[7] See Appendix D for full numeric tables and area under the curve (AUC)/probability of improvement plots.
[8] See Appendix F for ablation studies of L-AMIGo's grounding network and the components of L-NovelD.

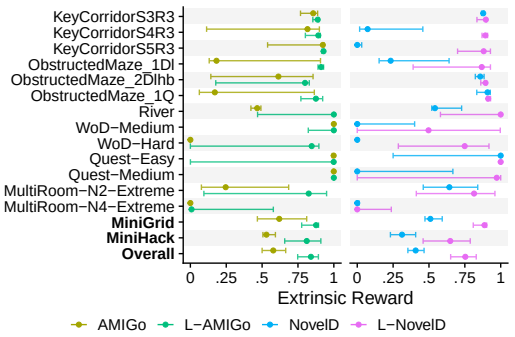


Figure 4: **Aggregate performance**. Interquartile mean (IQM) of models across tasks. Dot is median; error bars are 95% bootstrapped CIs.

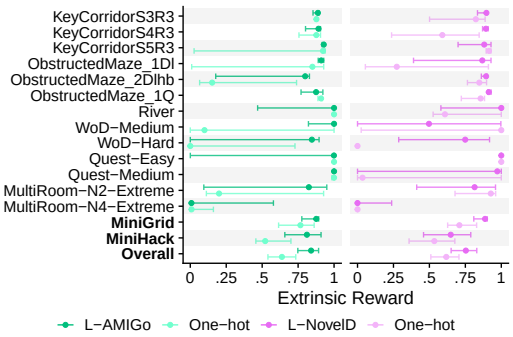


Figure 5: **One-hot performance.** Models compared to variants with one-hot non-compositional goals. Plot elements same as Figure 4

Linguistic exploration outperforms non-linguistic exploration. Both algorithms, L-AMIGo and L-NovelD, outperform their nonlinguistic counterparts. Despite variance in runs and across environments, averaged across all environments (**Overall**) we see a statistically significant improvement of L-AMIGo over AMIGo (.27 absolute, 47% relative) and of L-NovelD over NovelD (.35 absolute, 85% relative). In some tasks, Figure 3 shows that L-AMIGo and L-NovelD reach the same asymptotic performance as their non-linguistic versions, but with better sample efficiency and stability (e.g. KeyCorridorS3R3 L-AMIGo, Quest-Easy L-NovelD; see Appendix D.3 AUC plots). Lastly, the failure of the naive message reward shows that indiscriminate reward shaping fails in tasks with sufficiently diverse language; instead, some notion of novelty or difficulty is needed to make progress.

Linguistic exploration excels in larger environments. Our tasks include sequences of environments with the same underlying dynamics, but larger state spaces and thus more challenging exploration problems. In general, larger environments result in bigger improvements of linguistic over non-linguistic exploration, since the space of messages remains relatively constant even as the state space grows. For example, there is no difference in ultimate performance for language/non-language variants on KeyCorridorS3R3, yet the gaps grow as the environment size grows to KeyCorridorS5R3, especially in L-NovelD. A similar trend can be seen in the WoD tasks, where AMIGo actually outperforms L-AMIGo in WoD-Medium, but in WoD-Hard is unable to learn at all.

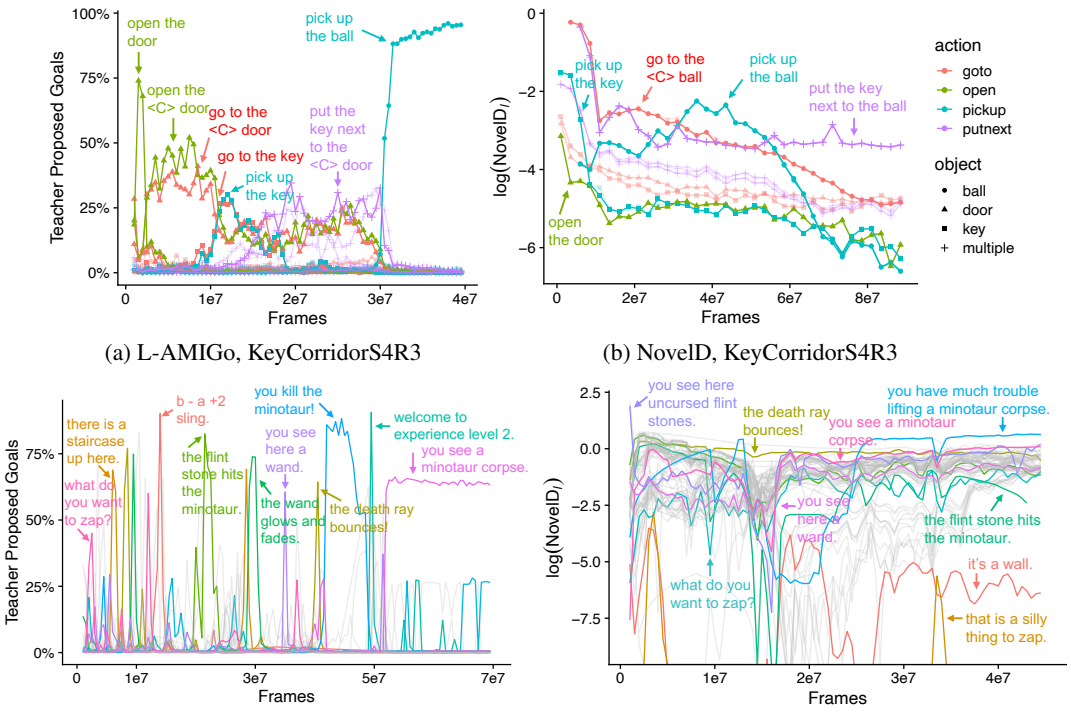
7.1 Interpretability

One auxiliary benefit of our language-based methods is that the language states and goals can provide insight into an agents' training and exploration process. We demonstrate how L-AMIGo and L-NovelD agents can be interpreted in Figure 6.

Emergent L-AMIGo Curricula. Campero et al. [7] showed that AMIGo teachers produce an interpretable curriculum, with initially easy (x, y) goals located next to the student's start location, and later harder goals referencing distant locations behind doors. In L-AMIGo, we can see a similar curriculum emerge through the proportion of *language goals* proposed by the teacher throughout training. In the KeyCorridorS4R3 environment (Figure 6a), the teacher first proposes the generic goal *open (any) door* before then proposing goals involving specific colored doors (where <C> is a color). Next, the agent discovers keys, and the teacher proposes *pick[ing] up the key* and putting it in certain locations. Finally, the teacher and student converge on the extrinsic goal *pick up the ball*.

Due to the complexity of the WoD-Hard environment, the curriculum for the teacher is more exploratory (Figure 6c). The teacher proposes useless goals at first, such as finding staircases and slings. At one point, the teacher proposes throwing stones at the minotaur (an ineffective strategy) before devoting more time towards wand actions (*you see here a wand*, *the wand glows and fades*). Eventually, as the student grows more competent, the teacher begins proposing goals that involve directly killing the minotaur (*you kill the minotaur*, *welcome to experience level 2*) before converging on the message *you see a minotaur corpse*—the final message needed to complete the episode.

L-NovelD Message Novelty. Similarly, L-NovelD allows for interpretation by examining the messages with highest intrinsic reward as training progresses. In KeyCorridorS4R3 (Figure 6b), the



(a) L-AMIGo, KeyCorridorS4R3

(b) NovelD, KeyCorridorS4R3

(c) L-AMIGo, WoD-Hard

(d) NovelD, WoD-Hard

Figure 6: **Interpretation of language-guided exploration.** For the KeyCorridorS4R3 and WoD-Hard environments, shown are curricula of goals proposed by the L-AMIGo teacher (a,c) and the intrinsic reward of messages (some examples labeled) for L-NovelD (b,d).

novelty of easy goals such as *open the door* decreases fastest, while the novelty of the true extrinsic goal (*pick up the ball*) and even rarer actions (*put the key next to the ball*) remains high throughout training. In WoD-Hard (Figure 6d), messages vary widely in novelty: simple and nonsensical messages like *that is a silly thing to zap* and *it's a wall* quickly plummet, while more novel messages are rarer states that require killing the minotaur (*you have trouble lifting a minotaur corpse*).

7.2 Do semantics matter?

Language not only denotes meaningful features in the world; its lexical and compositional semantics also explain how actions and states relate to each other. For example, in L-AMIGo, an agent might more easily *go to the red door* if it already knows how to *go to the yellow door*. Similarly, in L-NovelD, training the RND network on the message *go to the yellow door* could lower novelty of similar messages like *go to the red door*, which might encourage exploration of semantically broader states. While our primary focus is not on whether agents can generalize to new language instructions or states, we are still interested in whether these semantics improve exploration for extrinsic rewards.

To check this hypothesis, in Figure 5 we run "one-hot" variants of L-AMIGo and L-NovelD where the semantics of the language annotations are hidden: each message is replaced with a one-hot identifier (e.g. *go to the red door* $\rightarrow$ 1, *go to the blue door* $\rightarrow$ 2) but otherwise functions identically to the original message. We make two observations. **(1)** One-hot goals actually perform quite competitively, demonstrating that the primary benefit of language in these tasks is to abstract over the state space, rather than provide fine-grained semantic relations between states. **(2)** Nevertheless, L-AMIGo is able to exploit semantics, with a significant improvement (.20 absolute, 32% relative) in aggregate performance over one-hot goals, in contrast to L-NovelD, which shows no significant difference. We leave for future work a more in-depth investigation into what kinds of environments and models might benefit more from language semantics.

8 Discussion

The key insight in this paper is that language, even if noisy and often unrelated to the goal, is a more abstract, efficient, and interpretable space for exploration than state representations. To support this, we have presented variants of two popular state-of-the-art exploration methods, L-AMIGo and L-NovelD, that outperform their non-linguistic counterparts by 47–85% across 13 language-annotated tasks in the challenging MiniGrid and MiniHack environment suites.

Despite their success here, our models have some limitations. First, as is common in work like ours, it will be important to alleviate the restriction on oracle language annotations, perhaps by using learned state description models [31, 49]. Furthermore, L-AMIGo specifically cannot handle tasks such as the full NetHack game which have unbounded language spaces and many redundant goals (e.g. *go to/approach/arrive at the door*), since it selects a single goal which must be achieved verbatim. An exciting extension to L-AMIGo would propose abstract goals (e.g. *kill [any] monster* or *find a new item*), possibly in a continuous semantic space, that can be satisfied by multiple messages.

More general extensions include better understanding when and why language semantics benefits exploration (Section 7.2) and using pretrained models to imbue semantics into the models beforehand [49]. Additionally, although the agents in this work are able to explore even when not all language is useful, we must take caution in adversarial settings where the language is completely unrelated to the extrinsic task (and thus useless) or even describes harmful behaviors. Future work should measure how robust these methods are to the noisiness and quality of the language. Nevertheless, the success of L-AMIGo and L-NovelD demonstrates the power of even noisy language in these domains, underscoring the importance of abstract and semantically-meaningful measures of exploration in RL.

Acknowledgments and Disclosure of Funding

We thank Heinrich Küttler, Mikael Henaff, Andy Shih, Alex Tamkin, and anonymous reviewers for constructive comments and feedback, and Mikayel Samvelyan for help with MiniHack. JM is supported by an Open Philanthropy AI Fellowship.

References

[1] J. Achiam and S. Sastry. Surprise-based intrinsic motivation for deep reinforcement learning. *arXiv preprint arXiv:1703.01732*, 2017.

[2] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*, 2021.

[3] D. Bahdanau, F. Hill, J. Leike, E. Hughes, A. Hosseini, P. Kohli, and E. Grefenstette. Learning to understand goal specifications by modelling reward. In *International Conference on Learning Representations*, 2018.

[4] M. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos. Unifying count-based exploration and intrinsic motivation. In *Advances in Neural Information Processing Systems*, 2016.

[5] V. Blukis, Y. Terme, E. Niklasson, R. A. Knepper, and Y. Artzi. Learning to map natural language instructions to physical quadcopter control using simulated flight. In *Proceedings of the 3rd Conference on Robot Learning*, 2019.

[6] Y. Burda, H. Edwards, A. Storkey, and O. Klimov. Exploration by random network distillation. In *International Conference on Learning Representations*, 2018.

[7] A. Campero, R. Raileanu, H. Kuttler, J. B. Tenenbaum, T. Rocktäschel, and E. Grefenstette. Learning with AMIGo: Adversarially motivated intrinsic goals. In *International Conference on Learning Representations*, 2021.

[8] M. Chevalier-Boisvert, L. Willems, and S. Pal. Minimalistic gridworld environment for OpenAI gym. `https://github.com/maximecb/gym-minigrid`, 2018.

[9] M. Chevalier-Boisvert, D. Bahdanau, S. Lahlou, L. Willems, C. Saharia, T. H. Nguyen, and Y. Bengio. BabyAI: A platform to study the sample efficiency of grounded language learning. In *International Conference on Learning Representations*, 2019.

[10] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734, 2014.

[11] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). In *International Conference on Learning Representations*, 2016.

[12] C. Colas, A. Akakzia, P.-Y. Oudeyer, M. Chetouani, and O. Sigaud. Language-conditioned goal generation: a new approach to language grounding for RL. *arXiv preprint arXiv:2006.07043*, 2020.

[13] C. Colas, T. Karch, N. Lair, J.-M. Dussoux, C. Moulin-Frier, P. Dominey, and P.-Y. Oudeyer. Language as a cognitive tool to imagine goals in curiosity driven exploration. In *Advances in Neural Information Processing Systems*, 2020.

[14] C. Colas, T. Karch, O. Sigaud, and P.-Y. Oudeyer. Autotelic agents with intrinsically motivated goal-conditioned reinforcement learning: A short survey. *arXiv preprint arXiv:2012.09830*, 2020.

[15] M.-A. Côté, A. Kádár, X. Yuan, B. Kybartas, T. Barnes, E. Fine, J. Moore, M. Hausknecht, L. El Asri, M. Adada, W. Tay, and A. Trischler. TextWorld: A learning environment for text-based games. In *Workshop on Computer Games*, pages 41–75, 2018.

[16] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[17] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. IMPALA: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *Proceedings of the 35th International Conference on Machine Learning*, pages 1407–1416, 2018.

[18] M. Fang, T. Zhou, Y. Du, L. Han, and Z. Zhang. Curriculum-guided hindsight experience replay. In *Advances in Neural Information Processing Systems*, 2019.

[19] C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel. Reverse curriculum generation for reinforcement learning. In *Proceedings of the 1st Conference on Robot Learning*, pages 482–495, 2017.

[20] S. Forestier, R. Portelas, Y. Mollard, and P.-Y. Oudeyer. Intrinsically motivated goal exploration processes with automatic curriculum learning. *arXiv preprint arXiv:1708.02190*, 2017.

[21] J. Fu, A. Korattikara, S. Levine, and S. Guadarrama. From language to goals: Inverse reinforcement learning for vision-based instruction following. In *International Conference on Learning Representations*, 2019.

[22] P. Goyal, S. Niekum, and R. J. Mooney. Using natural language for reward shaping in reinforcement learning. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, 2019.

[23] P. Goyal, S. Niekum, and R. J. Mooney. Pixl2r: Guiding reinforcement learning using natural language by mapping pixels to rewards. In *Proceedings of the 4th Conference on Robot Learning*, pages 485–497, 2020.

[24] B. Harrison, U. Ehsan, and M. O. Riedl. Guiding reinforcement learning exploration using natural language. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1956–1958, 2018.

[25] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[26] Y. Jiang, S. Gu, K. Murphy, and C. Finn. Language as an abstraction for hierarchical deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2019.

[27] H. Küttler, N. Nardelli, T. Lavril, M. Selvatici, V. Sivakumar, T. Rocktäschel, and E. Grefenstette. TorchBeast: A PyTorch platform for distributed RL. *arXiv preprint arXiv:1910.03552*, 2019.

[28] H. Küttler, N. Nardelli, A. H. Miller, R. Raileanu, M. Selvatici, E. Grefenstette, and T. Rocktäschel. The NetHack learning environment. In *Advances in Neural Information Processing Systems*, 2020.

[29] M. C. Machado, M. G. Bellemare, and M. Bowling. Count-based exploration with the successor representation. In *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, pages 5125–5133, 2020.

[30] J. Martin, S. N. Sasikumar, T. Everitt, and M. Hutter. Count-based exploration in feature space for reinforcement learning. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence (IJCAI-17)*, pages 2471–2478, 2017.

[31] S. Mirchandani, S. Karamcheti, and D. Sadigh. ELLA: Exploration through learned language abstraction. In *Advances in Neural Information Processing Systems*, 2021.

[32] G. Ostrovski, M. G. Bellemare, A. Oord, and R. Munos. Count-based exploration with neural density models. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2721–2730, 2017.

[33] P.-Y. Oudeyer and F. Kaplan. How can we define intrinsic motivation? In *Proceedings of the 8th International Conference on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems*, 2008.

[34] P.-Y. Oudeyer and F. Kaplan. What is intrinsic motivation? A typology of computational approaches. *Frontiers in Neurorobotics*, 1:6, 2009.

[35] P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286, 2007.

[36] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell. Curiosity-driven exploration by self-supervised prediction. In *Proceedings of the 34th International Conference on Machine Learning*, pages 2778–2787, 2017.

[37] R. Portelas, C. Colas, K. Hofmann, and P.-Y. Oudeyer. Teacher algorithms for curriculum learning of deep RL in continuously parameterized environments. In *Proceedings of the 4th Conference on Robot Learning*, pages 835–853, 2020.

[38] R. Portelas, C. Colas, L. Weng, K. Hofmann, and P.-Y. Oudeyer. Automatic curriculum learning for deep RL: A short survey. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence (IJCAI-20) Survey Track*, pages 4819–4825, 2020.

[39] S. Racaniere, A. K. Lampinen, A. Santoro, D. P. Reichert, V. Firoiu, and T. P. Lillicrap. Automated curricula through setter-solver interactions. In *International Conference on Learning Representations*, 2020.

[40] R. Raileanu and T. Rocktäschel. RIDE: Rewarding impact-driven exploration for procedurally-generated environments. In *International Conference on Learning Representations*, 2020.

[41] M. Samvelyan, R. Kirk, V. Kurin, J. Parker-Holder, M. Jiang, E. Hambro, F. Petroni, H. Kuttler, E. Grefenstette, and T. Rocktäschel. MiniHack the planet: A sandbox for open-ended reinforcement learning research. In *Advances in Neural Information Processing Systems (Datasets and Benchmarks Track)*, 2021.

[42] J. Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In *Proceedings of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 222–227, 1991.

[43] J. Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

[44] E. Schwartz, G. Tennenholtz, C. Tessler, and S. Mannor. Language is power: Representing states using natural language in reinforcement learning. *arXiv preprint arXiv:1910.02789*, 2019.

[45] M. Shridhar, X. Yuan, M.-A. Côté, Y. Bisk, A. Trischler, and M. Hausknecht. ALFWorld: Aligning text and embodied environments for interactive learning. In *International Conference on Learning Representations*, 2021.

[46] B. C. Stadie, S. Levine, and P. Abbeel. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv preprint arXiv:1507.00814*, 2015.

[47] A. L. Strehl and M. L. Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008.

[48] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT Press, 1998.

[49] A. C. Tam, N. C. Rabinowitz, A. K. Lampinen, N. A. Roy, S. C. Chan, D. Strouse, J. X. Wang, A. Banino, and F. Hill. Semantic exploration from language abstractions and pretrained representations. *arXiv preprint arXiv:2204.05080*, 2022.

[50] H. Tang, R. Houthooft, D. Foote, A. Stooke, X. Chen, Y. Duan, J. Schulman, F. De Turck, and P. Abbeel. #exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2017.

[51] T. Tasrin, M. S. A. Nahian, H. Perera, and B. Harrison. Influencing reinforcement learning through natural language guidance. *arXiv preprint arXiv:2104.01506*, 2021.

[52] J. Urbanek, A. Fan, S. Karamcheti, S. Jain, S. Humeau, E. Dinan, T. Rocktäschel, D. Kiela, A. Szlam, and J. Weston. Learning to speak and act in a fantasy text adventure game. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 673–683, 2019.

[53] N. Waytowich, S. L. Barton, V. Lawhern, and G. Warnell. A narration-based reward shaping approach using grounded natural language commands. *arXiv preprint arXiv:1911.00497*, 2019.

[54] T. Zhang, H. Xu, X. Wang, Y. Wu, K. Keutzer, J. E. Gonzalez, and Y. Tian. NovelD: A simple yet effective exploration criterion. In *Advances in Neural Information Processing Systems*, 2021.

A L-AMIGo algorithm

Algorithm S1 Asynchronous learning step for L-AMIGo

Input: student batch size B_S, teacher policy batch size B_{policy}, grounding network batch size B_{ground}
$\mathcal{B}_{\text{policy}} \leftarrow \varnothing, \mathcal{B}_{\text{ground}} \leftarrow \varnothing$ ▷ Init teacher batches
$\mathcal{G} \leftarrow \varnothing$ ▷ Track descriptions seen thus far
while not converged **do**
    Sample batch $\mathcal{B}$ of size B_S from actors
    Train student on $\mathcal{B}$
    Add new descriptions ℓ_t in the batch to $\mathcal{G}$
    Update $\mathcal{B}_{\text{policy}}$ with (s_0, ℓ_t, r_t^T) tuples where goal ℓ_t completed ($r_t^T = +\alpha$) or episode ended ($r_t^T = -\beta$)
    if $|\mathcal{B}_{\text{policy}}| > B_{\text{policy}}$ **then**
        Train teacher policy on $\mathcal{B}_{\text{policy}}$; $\mathcal{B}_{\text{policy}} \leftarrow \varnothing$
    end if
    Update $\mathcal{B}_{\text{ground}}$ with (s_0, ℓ_{1st}) tuples
    if $|\mathcal{B}_{\text{ground}}| > B_{\text{ground}}$ **then**
        Train grounding net on $\mathcal{B}_{\text{ground}}$; $\mathcal{B}_{\text{ground}} \leftarrow \varnothing$
    end if
end while

Algorithm S1 describes the joint student/teacher learning step of L-AMIGo. Batches of experience $\mathcal{B}$ are generated from actors, which are used to update the student policy, and the teacher policy at different intervals defined by batch sizes B_{policy} and B_{ground}. The set of goals $\mathcal{G}$ known to the teacher is also progressively updated.

To reiterate the teacher training process: the teacher policy network is trained on tuples of student initial state s_0, proposed goal ℓ_t, and teacher rewards r_t^T (which is $+\alpha$ if the goal was completed by the student in $\geq t^*$ steps, or $-\beta$ if the goal was completed in $< t^*$ steps or never completed before episode termination). The teacher grounding network is trained on tuples of initial state s_0 and the first language description encountered along that trajectory ℓ_{1st}. Given s_0, the grounding network is asked to predict 1 for ℓ_{1st} and 0 for all other goals known to the teacher at the time.

B Details on naive message reward and equivalencies to prior work

As discussed in Section 2, prior approaches to language-guided reward shaping assume either a single extrinsic goal, or that language annotations are always helpful for progress in the environment. Here we show how these methods can all be approximately captured by a naive message reward, which our experiments show is insufficient for learning in sufficiently large linguistic spaces (Figure 3).

- Harrison et al. [24] (also [51]) propose a policy shaping method that generates language annotations for (action, state) pairs from simulated oracles, then for each language annotation learns a "critique policy" that mimics the human action distributions for the current state and language annotation. This is used to update the prior action distribution of the learned policy during training: given a state, the most likely language annotation over all possible actions is inferred (via Bayes' rule); then the critique policy for this language annotation is mixed in with the current policy's action distribution via a tunable hyperparameter. While the focus is on policy shaping rather than reward shaping, agents are simply pushed in the direction of the most salient language annotation, and there are no preferences for or against certain language. The overall effect, then, is that policies are pushed indiscriminately towards encountering *any* message in the environment, which is similar to a fixed message reward.

  A further limitation of Harrison et al. [24] that prevents straightforward application here is that it requires sufficient training data for training a critique policy offline *for every possible message encountered in the environment*. In contrast, we assume no such pretraining data; in fact, we have no knowledge about what messages we might encounter at all.

- ELLA [31] is closer to our setting, since the authors acknolwedge that not all messages or subgoals in a trajectory may be relevant for an overall goal, especially in the multi-task settings they explore where the extrinsic goal is subject to change. Accordingly, ELLA proposes to learn a "Relevance Classifier" that predicts whether or not a message is useful for the extrinsic goal by training on the messages encountered along trajectories that resulted in positive reward. However, in our setting, we have no extrinsic goals, and in most of our tasks, random exploration fails to attain *any* positive trajectories to provide such training data for a classifier (Figure 3, IMPALA curves). We could implement ELLA with an uninformative reference classifier, i.e. one that assumes all messages are relevant for the extrinsic goal. Then ELLA reduces to simply giving a fixed reward for any message encountered.
- Finally, a large class of reward shaping and inverse RL methods operate primarily by giving rewards associated with a (linguistic) extrinsic goal [3, 5, 21–24, 31, 44, 51, 53]. As one representative example, LEARN [22] proposes to train a model to associate an action at a given timestep with the probability it is *related* to an extrinsic language command: $p_R(a_t, \ell)$, as well as the complementary probability that an action is *unrelated*: $p_U(a_t, \ell) = 1 - p_R(a_t, \ell)$. The difference in these probabilities can thus be used as a reward signal: $r_t^i = p_R(a_t, \ell) - p_U(a_t, \ell)$.

  However, in our case we have no extrinsic instruction; rather, we have many intermediate low-level messages of unknown relevance to the extrinsic goal. A naive solution for LEARN-style approaches is thus to do reward shaping for every intermediate message, i.e. assign rewards whenever any annotation ℓ is deemed relevant. Moreover, we do not need to predict when an action is relevant to ℓ; the frequent annotations we receive already confirm that an action is linguistically relevant. Thus, in our setting, we can set $p_R(a, \ell) = 1$ if ℓ is observed in the given state, and $p_R(a, \ell) = 0$ otherwise.[9] Using this as the reward is equivalent to the baseline employed here. Technically, LEARN proposes a slightly modified reward $r_t'^i = \gamma r_t^i - r_{t-1}^i$, i.e. the difference in reward between timesteps where γ is the MDP discount factor, but this made no difference in our experiments; thus, for brevity we report the single fixed reward.

For the naive reward baseline reported in Figure 3, we performed a grid search on the intrinsic reward coefficient $\lambda \in \{0.1, 0.5, 1.0\}$, though modifying λ made no difference and results are reported with $\lambda = 0.1$. In all cases, agents trained with such rewards fail because without some notion of message novelty or difficulty, the agents are stuck exploiting easy-to-achieve, locally-optimal messages (e.g. running into the nearest wall in MiniHack, or the nearest door in MiniGrid); in fact, rewards for easy messages *discourage* exploration and lead to worse reward than even the vanilla IMPALA baselines! Thus, some way of measuring novelty/progress in the space of language annotations must be used, which is operationalized in L-AMIGo (via the continually growing difficulty threshold given to the teacher) and L-NovelD (via the decaying reward given for seeing the same message over and over again), though note that for L-NovelD, using a message reward with novelty-based decay is insufficient (Appendix F.2).

C Architecture and training details

Here, we describe the architecture, training details, and hyperparameters for the MiniGrid and MiniHack tasks. Our code is available in the supplementary material and also at `https://anonymized`.

C.1 MiniGrid

All models are adapted from Campero et al. [7]. Figure S1 from Campero et al. [7] details the architecture of the standard AMIGo student and teacher.

AMIGo student. The student gets an $7 \times 7 \times 3$ partial and egocentric grid representation, where each cell in the 7×7 grid is represented by 3 features indicating the type of the object, color of

[9] An even stricter interpretation would be to give partial rewards even for states with the null message $\varnothing$ by training a classifier to predict relevance for any non-null linguistic state. However, intermediate message rewards are already fairly common in our setting (it is easy to walk to the nearest wall in MiniHack, for example), and this would not solve the fundamental problem that without learning to disprefer certain messages, an agent will be stuck pursuing locally optimal messages.

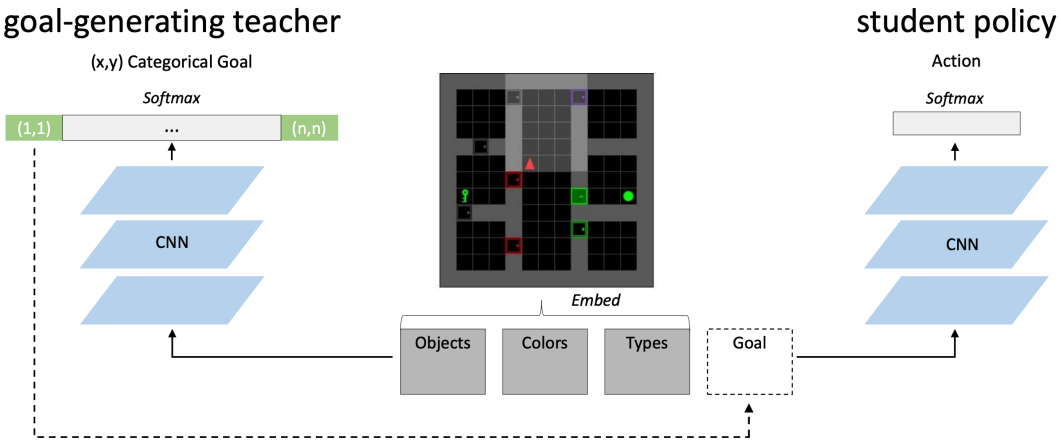


Figure S1: **Original AMIGo model overview.** Original figure from Campero et al. [7], reproduced with permission.

object, and object state (e.g. for doors, open/closed). The student first embeds the features into type/color/state embeddings of size 5, 3, and 2 respectively, as well as the (x, y) goal as a singleton feature, to get a $7 \times 7 \times 11$ grid which is then fed through a 5-layer convolutional neural network interleaved with Exponential Linear Units (ELUs; 11). Each layer has 32 output channels, filter size of $(3, 3)$, 2 stride, and 1 padding. The output of the ConvNet is then flattened and fed through a fully-connected layer to produce a 256-d state embedding. The policy and value functions are linear layers on top of this state embedding. The policy in particular produces a distribution over 7 possible actions (forward, pick up, put down, open, left, right, and a "done" action).

AMIGo teacher. The teacher gets an $N \times N \times 3$ fully observed grid encoding of the environment, where N varies according to the environment size. Like the student, the teacher embeds the grid representation into embeddings which are then fed through a 4-layer dimensionality-preserving neural network again interwoven with ELUs. Each convolutional layer has 16 output channels, filter size of $(3, 3)$, 1 stride, and 1 padding, except the last layer which has only 1 output channel. The ConvNet processes the input embeddings into a spatial action distribution over grid locations, from which an goal is sampled. The value head takes as input the penultimate layer from the ConvNet (i.e. a grid of $N \times N \times 16$) to produce the value estimate.

L-AMIGo Student. The L-AMIGo student has the same ConvNet base as the standard AMIGo student, but without an (x, y) goal channel feature (so the input to the ConvNet is $7 \times 7 \times 10$ instead of 11). To encode the goal, the student uses a one-layer Gated Recurrent Unit (GRU; Cho et al. [10]) recurrent neural network (RNN) with embedding size 64 and hidden size 256. The last hidden state of the GRU is taken as the goal representation, which is then concatenated with the state embedding to be fed into the policy and value functions, respectively.

L-AMIGo Teacher. The L-AMIGo *policy network* has the same ConvNet as the standard AMIGo teacher, but without the last layer (so the output is a $N \times N \times 16$ grid). The last output of the ConvNet is averaged across the spatial map to form a final 16-dimensional state embedding. The L-AMIGo teacher also has a GRU of same dimensionality as the L-AMIGo student. To propose a goal, the teacher embeds each known goal in the vocabulary with the GRU to produce 256-dimensional goal embeddings which are then projected via a linear layer into the 16-dimensional goal embedding space. The dot product of the goal embeddings and the state form the logits of the goal distribution. The value head takes as input the $N \times N \times 16$ unaveraged state representation concatenated with the logits of the distribution of language goals.

The *grounding network* also uses the 16-dimensional state embedding and the same GRU as the L-AMIGo policy network, but the 256-d goal embeddings are projected via a separate linear layer to a separate set of 16-dimensional embeddings. The dot product between these grounder-specific goal embeddings and the state embedding represent the log probabilities of the goal being achievable in an environment.

NovelD and L-NovelD. For NovelD experiments we use the student policy of L-AMIGo but with the goal embedding always set to the 0 vector.

The NovelD RND network uses the same convolutional network as the AMIGo/L-AMIGo students, taking in an egocentric agent view. The L-NovelD message embedding network uses a GRU parameterized identically to those of the L-AMIGo student and teacher. For NovelD experiments we set $\alpha = 1$, scale the RND loss by 0.1, and use the same learning rate as the main experiments. Using a grid search, we found optimal scaling factors of the standard NovelD reward to be 0.5 and the L-NovelD reward (i.e. λ_ℓ) also to be 0.5.

Hyperparameters. We use the same hyperparameters as in the original AMIGo paper: a starting difficulty threshold t^* of 7, a maximum difficulty t^* of 100, positive reward for the teacher $+\alpha = 0.7$, negative reward $-\beta = -0.3$, learning rate 10^{-4} which is linearly annealed to 0 throughout training, batch size 32, teacher policy batch size 32, teacher grounder batch size 100, unroll length 100, RMSprop optimizer with $\varepsilon = 0.01$ and momentum 0, entropy cost 0.0005, generator entropy cost 0.05, value loss cost 0.5, intrinsic reward coefficient $\lambda = 1$.

C.2 MiniHack

Models are adapted from baselines established for the NetHack [28] and MiniHack [41] Learning Environments.

AMIGo student. The student is a recurrent LSTM-based [25] policy. The NetHack observation contains a 21×79 matrix of glyph identifiers, a 21-dimensional feature vector of agent stats (e.g. position, health, etc), and a 256-character (optional) message. The student produces 4 dense representations from this observation which are then concatenated to form the state representation.

First, an embedding of the entire game area is created. Each glyph is converted into a 64-dimensional vector embedding, i.e. the input is now a $21 \times 79 \times 64$ grid. This entire grid is fed through a 5-layer ConvNet interleaved with ELUs. Each conv layer has a filter size of $(3, 3)$, stride 1, padding 1, and 16 output channels, except for the last which has 8 output channels. **Second**, an embedding of a 9×9 egocentric crop of the grid around the agent is created. This is created by feeding the crop through another separately-parameterized ConvNet of the same architecture. **Third**, the 21-d feature vector of agent stats is fed into a 2-layer MLP (2 layers with 64-d outputs with ReLUs in between) to produce a 64-d representation of the agent stats. **Fourth**, the message is parsed with the BERT [16] WordPiece tokenizer and fed into a GRU with embedding size 64 and hidden size 256. These four embeddings are then concatenated and form the state representation, which is then fed into the LSTM which contextualizes the current representation. This final representation is fed into linear policy and value heads to produce action distribution and value estimates.

AMIGo teacher. The teacher first embeds the agent stats using the same MLP that the student uses. It has the same 5-layer ConvNet used by the student to embed the full game area, except the input channels are modified to accept the 64-d feature vector concatenated to every cell in the 21×79 grid, so the input to the ConvNet is $21 \times 79 \times 128$. Additionally, the output layer has only 1 output channel. This produces a spatial action distribution over the 21×79 grid from which a "goal" as (x, y) coordinate is sampled.

L-AMIGo student. The student works identically to the standard AMIGo student, except without the teacher goal channel concatenated into the grid input. Also, in addition to the 4 representation constructed from the observation, the student gets 2 more representations constructed from the teacher's language goal. **First**, the student uses the same GRU used to process the game message to encode the teacher's language goal to create a 256-d representation. **Second**, the difference between the observed language embedding and the teacher language embedding is used as an additional feature (when this is the 0 vector then the student has reached the goal). All together, this forms 6 representations that together constitute the state representation.

L-AMIGo teacher. The teacher constructs the same state representation as the standard AMIGo student (*not* the AMIGo teacher). The teacher then embeds all known goals with a word-level GRU with the same architecture as the message network used by the student. These 256-d message embeddings are then projected to the state representation hidden size, and the dot product between

message embeddings and state representations forms the distribution over goals. Similarly, for the grounding network, the 256-d message embeddings are projected via a separate linear layer to produce probabilities of achievability. Like in MiniGrid, the state representation and the goal logits are used as input for the value head.

NovelD and L-NovelD. As in MiniGrid experiments, for NovelD experiments we use the L-AMIGo student policy where the goal embedding is always 0.

The NovelD RND network uses the cropped ConvNet representation of the student, fed through a linear layer to produce a 256-d state representation. Egocentric crops change more over time and are thus a more reliable signal of "novelty" than the full grid representation (verified with experiments). Additionally, as done in Burda et al. [6], two more additional layers are added to the final MLP of the predictor network only (not the random target network). The L-NovelD message RND network uses the same word-level GRU architecture of the student. We set $\alpha = 0.5$, scale the RND loss by 0.1, and use the same learning rate as the main student policy. The standard NovelD reward is left alone, and the L-NovelD reward hyperparameter (λ_ℓ) is grid-searched and set to 50 for all MiniHack experiments except Quest-{Easy,Medium}, where it is 30.

Hyperparameters. We generally stick to the same hyperparameters of Samvelyan et al. [41]. We use a starting difficulty threshold t^* of 1, a maximum difficulty t^* of 2 (a high goal difficulty is not as important for MiniHack), positive reward for the teacher $+\alpha = 0.7$, negative reward $-\beta = -0.3$, linearly-annealed learning rate 10^{-4}, batch size 32, teacher policy batch size 32, teacher grounder batch size 500, unroll length 100, RMSprop optimizer with $\varepsilon = 0.01$ and momentum 0, entropy cost 0.0005, generator entropy cost 0.05, value loss cost 0.5, intrinsic reward coefficient $\lambda = 0.4$. ε-greedy exploration was used for the teacher policy with $\varepsilon = 0.05$, which we found helped learning.

C.3 Compute Details

Each model was run for 5 independent seeds on a machine in an independent cluster with 40 CPUs, 1 Tesla V100 GPU, and 64GB RAM. Runs take between 4 hours (for ObstructedMaze_1Dl) to 20 hours (for the longest MultiRoom-N4-Extreme and KeyCorridorS5R3 tasks).

D Additional tables visualizations of main results

D.1 Full numeric tables

Table S1 contains full numbers for IQM performance. This is the same data as Figure 4, just summarized in numeric form.

Table S1: **Full IQM numbers.** IQM performance ($\pm$ 95% bootstrapped CIs) for models across tasks.

Environment	AMIGo	L-AMIGo	NovelD	L-NovelD
KeyCorridorS3R3	0.86 (0.77, 0.89)	0.89 (0.85, 0.90)	0.88 (0.87, 0.88)	**0.90** (0.83, 0.90)
KeyCorridorS4R3	0.82 (0.11, 0.90)	**0.89** (0.80, 0.91)	0.07 (0.02, 0.45)	**0.89** (0.87, 0.91)
KeyCorridorS5R3	0.92 (0.54, 0.93)	**0.93** (0.92, 0.93)	0.00 (0.00, 0.03)	0.88 (0.70, 0.93)
ObstructedMaze_1Dl	0.18 (0.13, 0.91)	**0.91** (0.89, 0.93)	0.23 (0.15, 0.64)	0.87 (0.39, 0.93)
ObstructedMaze_2Dlhb	0.61 (0.14, 0.86)	0.80 (0.18, 0.83)	0.86 (0.82, 0.88)	**0.89** (0.86, 0.90)
ObstructedMaze_1Q	0.17 (0.06, 0.86)	0.88 (0.77, 0.92)	**0.91** (0.83, 0.93)	**0.91** (0.91, 0.93)
River	0.47 (0.42, 0.49)	**1.00** (0.47, 1.00)	0.54 (0.52, 0.73)	**1.00** (0.58, 1.00)
WoD-Medium	**1.00** (1.00, 1.00)	**1.00** (0.82, 1.00)	0.00 (0.00, 0.40)	0.50 (0.00, 1.00)
WoD-Hard	0.00 (0.00, 0.00)	**0.85** (0.00, 0.90)	0.00 (0.00, 0.00)	0.75 (0.29, 0.92)
Quest-Easy	**1.00** (0.99, 1.00)	**1.00** (0.00, 1.00)	**1.00** (0.25, 1.00)	**1.00** (1.00, 1.00)
Quest-Medium	**1.00** (0.99, 1.00)	**1.00** (1.00, 1.00)	0.00 (0.00, 0.67)	0.97 (0.00, 1.00)
MultiRoom-N2-Extreme	0.25 (0.08, 0.69)	**0.82** (0.09, 0.95)	0.64 (0.46, 0.84)	0.81 (0.41, 0.96)
MultiRoom-N4-Extreme	0.00 (0.00, 0.00)	**0.01** (0.01, 0.58)	0.00 (0.00, 0.01)	0.00 (0.00, 0.24)
MiniGrid	0.62 (0.47, 0.81)	0.88 (0.78, 0.89)	0.51 (0.47, 0.59)	**0.89** (0.81, 0.90)
MiniHack	0.53 (0.51, 0.59)	**0.81** (0.65, 0.91)	0.31 (0.23, 0.41)	0.65 (0.46, 0.79)
Overall	0.57 (0.50, 0.66)	**0.84** (0.74, 0.89)	0.41 (0.35, 0.47)	0.76 (0.66, 0.83)

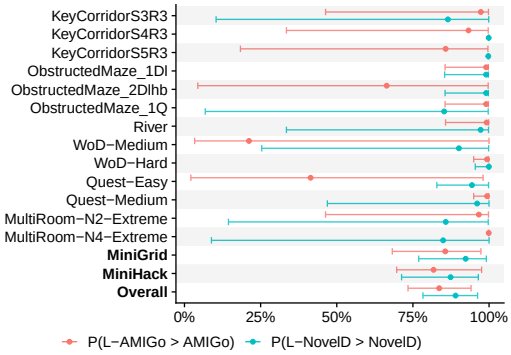


Figure S2: **Probability of improvement.** Probability of improvement of L-AMIGo over AMIGo, and L-NovelD over NovelD, as measured by Mann-Whitney U tests between their final performances. Plot elements same as Figure 4.

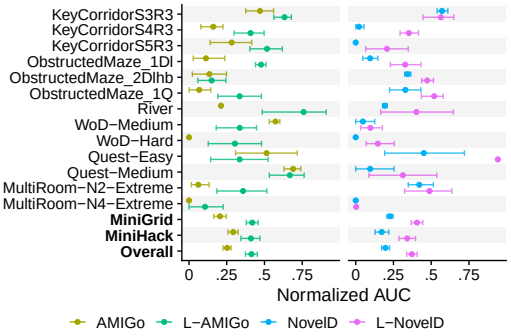


Figure S3: **Normalized AUC.** Normalized area under the curve (AUC) for AMIGo, L-AMIGo, NovelD, and L-NovelD. Plot elements same as Figure 4.

D.2 Probability of improvement

In addition to the IQM, another alternative interpretation of results as advocated in Agarwal et al. [2] is to use the *probability of improvement* of algorithm A over algorithm B, as measured by the nonparametric Mann-Whitney U test between independent runs from both algorithms. Figure S2 shows results from such an evaluation, again evaluated over bootstrapped confidence intervals constructed from 5000 samples per model/env combination. Qualitative results are the same as in Figure 4: overall, across environments, L-AMIGo and L-NovelD are both highly likely to outperform AMIGo and NovelD.

D.3 Area Under the Curve (AUC).

The IQM (Figure 4) and probability plots (Figure S2) use point estimates of ultimate performance attained after a fixed compute budget, which does not measure differences in sample efficiency between runs. As a measure which also elucidates differences in sample efficiency, we to use the normalized Area Under the Curve (AUC) to compare runs, as used in prior work [22]. Results are in Figure S3, and are qualitatively similar to the IQM and probability plots, but here, differences in sample efficiency can be seen in some environments, e.g. in KeyCorridorS3R3 for L-AMIGo, which are absent from the IQM plot.

E MiniGrid experiments with language encoded into the state representation

Here we run MiniGrid models with language encoded into the state representation. Full training curves for MiniGrid tasks only are located in Figure S4, IQM summary statistics for all environments (with only MiniGrid environments changed) are located in Figure S5, and Table S2 contains updated raw performance numbers. As discussed in the main text, recall that MiniHack models all already encode language into the state representation. We make the following observations about how these experiments differ from those in the main text:

Incorporating language into the feature space improves performance across all models. However, just incorporating language in the feature space is insufficient for competitive performance alone: while the IMPALA baseline is able to make more progress in simpler environments (e.g. now nearly solving KeyCorridorS3R3), it still lags behind and is unable to solve the harder environments. This clearly illustrates that in order to reap the benefits of language, it is important to use it in conjunction with exploration, not just as a feature.

L-AMIGo continues to outperform AMIGo, though the differences are smaller. While the tables show that both L-AMIGo and AMIGo can solve each task, reaching roughly similar final performance, they differ in sample efficiency as well as training stability (Figure S4, as well as the

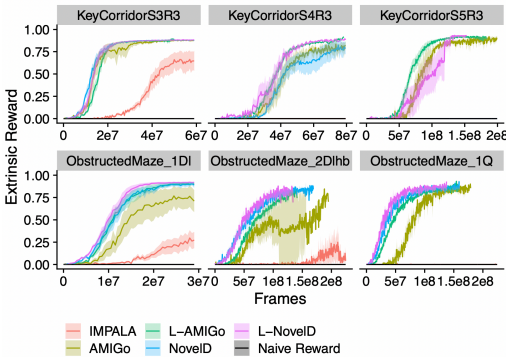


Figure S4: **Training curves for MiniGrid with language states**. Plot elements same as Figure 3.

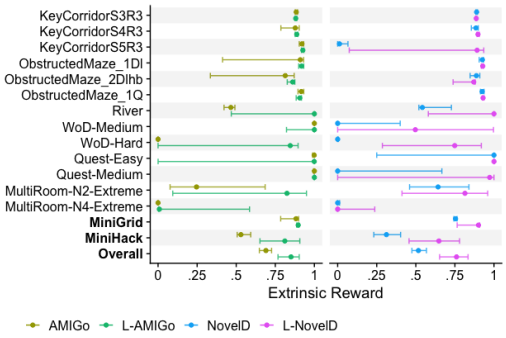


Figure S5: **IQMs with language states**. Plot elements same as Figure 4.

smaller error bars of L-AMIGo in Figure S5). Similar to the results in the main text, the full training curves in Figure S4 show that while L-AMIGo and AMIGo reach similar asymptotic performance, L-AMIGo learns quicker or more stably. This happens quite clearly in KeyCorridorS5R3 and the Maze environments.

L-NovelD and NovelD performance on MiniGrid are similar. This is the case except on a few tasks: KeyCorridorS4R3, and KeyCorridorS5R3 where NovelD is still unable to learn. Note that these NovelD results are somewhat unsurprising given the L-NovelD ablations and discussion in Appendix F.2; as we discuss there, L-NovelD is simply an adaptation of NovelD that enables a more precise tradeoff between language and state-based novelty. Similarly to how L-NovelD did not significantly outperform NovelD with language in the RND representation in Appendix F.2, L-NovelD does not outperform NovelD here, showing that for MiniGrid it is sufficient to naively combine the language and state representations. However, the MiniHack results in the main text indicate that there are settings where it is beneficial to have the separate L-NovelD term, and insufficient to solely encode language into the state representation.

To conclude, while adding language to the state representation results in smaller and more subtle performance differences in MiniGrid environments, the main conclusion (that language variants outperform their non-linguistic baselines) remains unchanged. As the last row of Table S2 shows, overall, L-AMIGo outperforms AMIGo by 23% (.16 absolute), and L-NovelD outperforms NovelD by 46% (.24 absolute) across environments; both differences remain statistically significant.

Table S2: **IQM numbers for MiniGrid with language states.** Table elements same as Table S1.

Environment	AMIGo	L-AMIGo	NovelD	L-NovelD
KeyCorridorS3R3	0.88 (0.88, 0.89)	0.88 (0.88, 0.89)	**0.89 (0.88, 0.89)**	**0.89 (0.88, 0.89)**
KeyCorridorS4R3	0.88 (0.79, 0.90)	0.89 (0.88, 0.89)	0.89 (0.85, 0.90)	**0.90 (0.89, 0.91)**
KeyCorridorS5R3	0.92 (0.90, 0.93)	**0.93 (0.92, 0.93)**	0.01 (0.00, 0.07)	0.89 (0.07, 0.93)
ObstructedMaze_1Dl	0.91 (0.41, 0.93)	0.92 (0.90, 0.93)	**0.93 (0.91, 0.93)**	**0.93 (0.92, 0.93)**
ObstructedMaze_2Dlhb	0.81 (0.34, 0.87)	0.86 (0.83, 0.87)	**0.89 (0.85, 0.91)**	0.87 (0.74, 0.88)
ObstructedMaze_1Q	0.92 (0.90, 0.93)	0.91 (0.88, 0.92)	0.92 (0.91, 0.94)	**0.93 (0.93, 0.94)**
MiniGrid	0.88 (0.79, 0.90)	**0.90 (0.89, 0.90)**	0.75 (0.74, 0.76)	**0.90 (0.76, 0.91)**
Overall (w/ MiniHack)	0.69 (0.65, 0.73)	**0.85 (0.77, 0.90)**	0.52 (0.47, 0.57)	**0.76 (0.65, 0.83)**

F Ablations

F.1 L-AMIGo grounding network

Figure S6 shows performance of L-AMIGo without the grounding network, where the policy network directly produces a distribution over goals without first predicting goal achievability, aggregated across

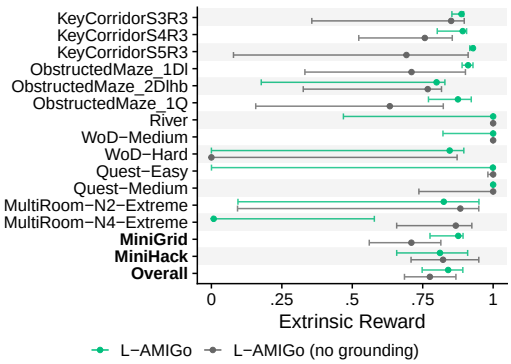


Figure S6: **L-AMIGo grounding network ablation**. IQM of L-AMIGo with and without grounding network across environments. Plot elements same as Figure 4.

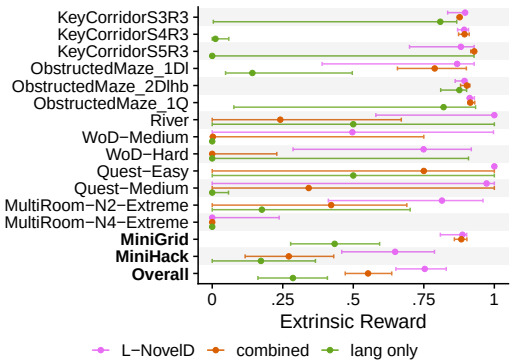


Figure S7: **L-NovelD ablations**. IQMs for Full L-NovelD, NovelD with combined state/language input, and language-only L-NovelD across environments. Plot elements same as Figure 4.

domains. Overall, L-AMIGo without the grounding network performs reasonably well, matching full L-AMIGo performance on MiniHack. On MiniGrid, we see a modest difference in aggregate performance, though importantly, we also see greatly increased training stability with the grounding network on individual environments. Specifically, the confidence intervals are much larger for L-AMIGo without the grounding network on **KeyCorridorS{3,4,5}R3** and **ObstructedMaze_{1Dl,1Q}**.

We hypothesize that the difference between MiniGrid and MiniHack tasks is because MiniGrid goals differ more between episodes: for example, since the colors of the doors are randomly shuffled, not all environments have red doors, so it is helpful to explicitly predict such features of the environment with a grounding network. In contrast, MiniHack envs have a more consistent set of goals (e.g. every WoD seed has a wand, a minotaur, etc), and so the grounding network is less necessary in this case.

F.2 L-NovelD components

To examine the relative performance of each component of L-NovelD, we run ablation experiments by (1) using the language-based reward *only*, or (2) combining the language and state embedding into a single representation. As discussed in Appendix B. using the language-based reward only is equivalent to the naive message baseline (and thus prior work like [22, 24, 31]) with an RND-determined novelty-based decay.

Results are in Figure S7. They show that using the language reward alone results in uniformly worse performance across environments, suggesting that a naive message-based reward, even with novelty decay, underperforms and it is important to provide a simpler navigation-based bonus to encourage exploration.[10] Additionally, while combining the state and language into a single embedding works well for MiniGrid, it does not work as well for MiniHack tasks, suggesting that the additional flexibility afforded by the separate L-NovelD term can be helpful in many settings. In principle, it should be possible to tune the state and language embedding sizes of combined NovelD to see comparable performance to L-NovelD, but the point of L-NovelD is to clarify the contributions made by both the language and state and make it easier to trade-off between the two.

G Full description of tasks and language

Here we describe each task in MiniGrid and MiniHack in detail, and enumerate the list of messages available in each task. Examples of all tasks explored in this work are located in Figure S9.

G.1 MiniGrid

Language. As described in Section 6.1, the set of possible descriptions in the BabyAI language 652 involves *goto, open, pickup*, and *putnext* commands which can be applied to boxes, doors, and

[10]Note also that message-only NovelD underperforms L-AMIGo as well.

```
go to the <C> ball
go to the <C> box
go to the <C> door
go to the <C> key
go to the ball
go to the box
go to the door
go to the key
open the <C> box
open the <C> door
open the box
open the door
pick up the <C> ball
pick up the <C> box
pick up the <C> key
pick up the ball
pick up the box
pick up the key
put the <C> ball next to the <C> ball
put the <C> ball next to the <C> box
put the <C> ball next to the <C> door
put the <C> ball next to the <C> key
put the <C> ball next to the ball
put the <C> ball next to the box
put the <C> ball next to the door
put the <C> ball next to the key
put the <C> box next to the <C> ball
put the <C> box next to the <C> box
put the <C> box next to the <C> door
put the <C> box next to the <C> key
put the <C> box next to the ball
put the <C> box next to the box
put the <C> box next to the door
put the <C> box next to the key
put the <C> key next to the <C> ball
put the <C> key next to the <C> box
put the <C> key next to the <C> door
put the <C> key next to the <C> key
put the <C> key next to the ball
put the <C> key next to the box
put the <C> key next to the door
put the <C> key next to the key
put the ball next to the <C> ball
put the ball next to the <C> box
put the ball next to the <C> door
put the ball next to the <C> key
put the ball next to the ball
put the ball next to the box
put the ball next to the door
put the ball next to the key
put the box next to the <C> ball
put the box next to the <C> box
put the box next to the <C> door
put the box next to the <C> key
put the box next to the ball
put the box next to the box
put the box next to the door
put the box next to the key
put the key next to the <C> ball
put the key next to the <C> box
put the key next to the <C> door
put the key next to the <C> key
put the key next to the ball
put the key next to the box
put the key next to the door
put the key next to the key
```

Figure S8: **Full list of possible MiniGrid messages**. Messages are synthesized with the BabyAI [9] grammar, then divided into 66 templates. <C> is one of 6 possible colors: *grey*, *green*, *blue*, *red*, *purple*, *yellow*.

balls, optionally qualified by color. These messages can be grouped into 66 message "templates" where specific colors are replaced with a placeholder <C>, as shown in Figure S8. Not all messages are needed for success on MiniGrid tasks, nor achieved by expert policies during training. We explain which messages are needed and/or encountered for each task below.

KeyCorridorS{3,4,5}R3. In these tasks (Figure S9a–c), the agent is tasked with picking up a ball behind a locked door. To do so, it must first find and pick up the key which is hidden in (possibly several) nested rooms, return to the locked door, unlock the door, place the key down, and pick up the ball. The agent start location, object colors, and room/door positions are randomized across seeds.

A typical policy trained on each task will encounter anywhere from 92 (KeyCorridorS3R3) to 141 messages (KeyCorridorS{4,5}R3) throughout training. Regardless of the environment size, a successful trajectory requires encountering anywhere from 8–12 messages: *go to the door* (up to 3x), *open the door* (up to 3x), *go to the key*, *pick up the key*, *go to the* [locked] *door*, *open the* [locked] *door*, *go to the ball*, *pick up the ball*.

ObstructedMaze-{1Dl,2Dlhb,1Q}. In these tasks (Figure S9d–f), the agent is also tasked with picking up a ball behind a locked door. In the easier 1Dl task, the key is in the open; in the harder tasks, the keys are hidden in boxes which must be opened, and the doors are blocked by balls which must be moved to access them.

A typical policy trained on each task will encounter anywhere from 66 messages (ObstructedMaze-1Dl) to 244 messages (ObstructedMaze-1Q) throughout training. Of these, a successful trajectory requires anywhere from 6 messages in ObstructedMaze-1Dl (*go to the key*, *pick up the key*, *go to the door*, *open the door*, *go to the ball*, *pick up the ball*) to 11 messages in ObstructedMaze-1Q (*go to the door*, *open the door*, *go to the box*, *open the box*, *pick up the key*, *go to the ball*, *pick up the ball*, *go to the door*, *open the door*, *go to the ball*, *pick up the ball*).

G.2 MiniHack

Language. As discussed previously, it is difficult to enumerate all messages in MiniHack, though we describe the messages encountered for each environment below, with a raw dump in Appendix H. We perform some preprocessing on the messages to prevent unbounded growth: we replace all numbers (e.g. *3 flint stones*, *2 flint stones*, etc) with a single variable *N*; we fix wands encountered in the environment to be a single Wand of Death (instead of having a wand of over 30 different types); items that can be arbitrarily named with random strings (e.g. *a scroll labeled zelgo mer*; *a dog named Hachi* have their names removed; finally, we cap the number of unique messages for each environment at 100.

River. In this task (Figure S9g), the agent must cross the river located to the right on the environment, which can only be done by planning and pushing at least two boulders into the river in a row (to form a bridge over the river). A typical policy will encounter 14 distinct messages during training (Appendix H.1), of which 7–8 messages (3–4 unique) are seen during training (*with great effort you move the boulder*, *you push the boulder into the water.*, *now you can cross it!*; these messages must be repeated twice).

Wand of Death ({Medium,Hard}). In these tasks (Figure S9i–j; also described in the main text), the agent must learn to use a *wand of death*, which can zap and kill enemies. This involves a complex sequence of actions: the agent must find the wand, pick it up, choose to *zap* an item, select the wand in the inventory, and finally choose the direction to zap (towards the minotaur which is pursuing the player). It must then proceed past the minotaur to the goal to receive reward. Taking these actions out of order (e.g. trying to *zap* something with nothing in the inventory, or selecting something other than the wand) has no effect.

In the Medium environment, the agent is placed in a narrow corridor with the wand somewhere in the corridor, and the minotaur is asleep (which gives the agent more time to explore). In the Hard environment, the agent is placed in a larger room where it must first find the wand, with the added challenge that the minotaur is awake and pursues the player, leading to death if the minotaur ever touches the player.

In both Medium and Hard environments, a policy encounters around 60 messages (Appendix H.2), of which 7 messages (7 unique) are typically necessary to complete the task (*you see here a wand*, *f - a wand*, *what do you want to zap?*, *in what direction? you klil the minotaur!*, *welcome to experience level 2*, *you see here a minotaur corpse*).

Quest ({Easy,Medium}). These tasks (Figure S9j–k) require learning to use a *Wand of Cold* to navigate over a river of lava while simultaneously fighting monsters. The agent spawns with a Wand of Cold in its inventory, and must learn to zap the Wand of Cold at the lava, which freezes it and forms a bridge over the lava.

In the Easy environment, the agent must first cross the lava river, then survive fights with one or two monsters to the staircase at the end of the hall. In the Medium environment, the agent must first fight several monsters *before* crossing the lava river. There is an additional challenge: note the narrow corridor before the main room in the Medium environment. If the agent runs into the room and tries to fight the monsters all at once, it quickly will become overwhelmed and die; to successfully kill all monsters, the agent must learn to use the narrow corridor as a "bottleneck", first baiting then leading the monsters into the corridor, until all are defeated. It can then use the Wand of Cold to cross the lava bridge to the staircase beyond.

In both Easy and Medium environments, a typical policy encounters the maximum of 100 messages (Appendix H.3). Very few messages, besides *what do you want to zap? / in what direction ? / the lava cools and solidifies* are *required* to complete an episode, since there is a large variety of monsters faced and uncertainty in their behaviors, which can trigger additional messages. However, highly efficient expert play typically encounters 8–12 messages (7–10 unique) in the Easy environment and 30–40 messages (8–12 unique) in the Medium environment: besides freezing the lava to cross the environment, the rest of the messages are combat related (e.g. *you kill the enemy!*, *an enemy corpse*, additional zapping commands, etc).

MultiRoom-N{2,4}-Extreme. These tasks (Figure S9l–m) are ported from the MiniGrid Multi-Room tasks, but with significant additional challenges. The ultimate task is to reach the last room in a sequence of interconnected rooms, but in the Extreme versions of this task, the walls are replaced with lava (which result in instant death if touched), there are monsters in each room (which must be fought), and additionally the doors are locked and must be "kicked" open (which requires that select the kick action, then kick in the direction of the door).

In both environments, a typical policy encounters the maximum of 100 messages (Appendix H.4). In a successful trajectory, an agent encounters 20–30 (12–14 unique) messages in MultiRoom-N2 and 60–70 messages (16–18 unique) in MultiRoom-N4. These messages are mostly combat-related (*the enemy hits!*, *you hit the enemy!*, *you kill the enemy*, with repeated messages relating to kicking down

F.1. River

```
the stairs are solidly fixed to the floor.
with great effort you move the boulder.
that was close.
it's solid stone.
you don't have anything to zap.
you try to move the border, but in vain.
you try to crawl out of the water.
there is a boulder here, but you cannot lift any more.
there is nothing here to pick up.
never mind.
pheew!
you push the boulder into the water.
now you can cross it!
it sinks without a trace!
```

F.2. WoD-{Medium,Hard}

```
c - a uncursed flint stone ( in quiver pouch ).
c - n uncursed flint stone ( in quiver pouch ).
in what direction?
it's a wall.
you don't have anything to zap.
you see here a wand.
you see here a uncursed flint stone.
you see here n uncursed flint stones.
there is nothing here to pick up.
what a strange direction!
the stairs are solidly fixed to the floor.
there is a staircase up here.
f - a wand.
the death ray bounces!
that is a silly thing to zap.
what do you want to throw?
what do you want to zap?
the wand glows and fades.
the death ray whizzes by you!
nothing happens.
b - a + 2 sling.
the wand turns to dust.
b - a blessed + 2 sling.
the flint stone misses the minotaur.
a wand shatters into a thousand pieces!
c - n uncursed flint stones.
c - a uncursed flint stone.
the flint stone hits another object.
the flint stone hits another object and falls down the stairs.
you see here a minotaur corpse.
welcome to experience level 2.
the flint stone hits the minotaur.
the flint stone hits other objects.
continue? ( q )
from the impact, the other object falls.
movement is very hard.
you stagger under your heavy load.
you kill the minotaur!
the sling misses the minotaur.
you have much trouble lifting a minotaur corpse.
the flint stone falls down the stairs.
the flint stone hits the minotaur!
the death ray misses the minotaur.
you see here a + 2 sling.
from the impact, the other objects fall.
the minotaur butts!
the sling hits the minotaur!
the sling hits the minotaur.
the wand misses the minotaur.
the wand falls down the stairs.
the sling hits another object.
g - a minotaur corpse.
you have extreme difficulty lifting a minotaur corpse.
the wand hits another object.
you see here a blessed + 2 sling.
the minotaur hits!
the flint stone hits other objects and falls down the stairs.
the sling hits another object and falls down the stairs.
the wand hits another object and falls down the stairs.
your movements are now unencumbered.
your movements are slowed slightly because of your load.
you can barely move a handspan with this load!
```

F.3. Quest-{Easy,Medium}

```
a enemy blocks your path.
a crude dagger misses you.
c - a uncursed flint stone ( in quiver pouch ).
f - a horn.
g - a wand.
h - a enemy corpse.
h - a crude dagger.
h - a scroll.
h - n darts.
i - a gem.
the wand glows and fades.
the enemy just misses!
the enemy hits!
the enemy picks up a uncursed flint stone.
the enemy throws a crude dagger!
the enemy touches you!
the enemy bites!
the enemy wields a crude dagger!
the enemy misses!
the enemy thrusts her crude dagger.
the stairs are solidly fixed to the floor.
the bolt of cold bounces!
the flint stone misses the enemy
the lava cools and solidifies.
in what direction?
that is a silly thing to zap.
it's a wall.
you are almost hit by a crude dagger.
you are hit by a crude dagger.
you get zapped!
you stop at the edge of the lava.
you kill the enemy
you miss the enemy
you destroy the enemy
there is nothing here to pick up.
what a strange direction!
what do you want to zap?
your movements are slowed slightly because of your load.
invalid direction for'g'prefix.
h - a wand.
h - n throwing stars.
i - a potion.
the enemy throws a dart!
the enemy escapes the dungeon!
the bolt of cold hits you!
the flint stone hits the enemy
you are hit by a dart.
a lit field surrounds you!
h - a skull cap.
h - a food ration.
h - a towel.
h - a dart.
i - a food ration.
i - a scroll.
i - n darts.
the bolt of cold hits the enemy
the dart misses the enemy
j - a dart.
you are almost hit by a dart.
you cannot escape from the enemy
```

doors (*in what direction?*, *as you kick the door, it crashes open*, *WHAAAAM!*, sometimes repeated up to 3 times per door, of which there are 1–3 doors).

H Raw MiniHack Messages

Messages are located on the next page. Each shows the list of messages encountered by a single agent trained on an environment in the corresponding categories. Separate training runs will encounter different messages.

```
a dart misses you.
h - a potion.
the enemy is killed!
c - n uncursed flint stone ( in quiver pouch ).
i - a wand.
the wand turns to dust.
the crude dagger hits the enemy
the crude dagger slips as the enemy throws it!
you don't have enough stamina to move.
you don't have anything to zap.
you rebalance your load.
nothing happens.
movement is difficult.
i - a conical hat.
you pull free from the enemy
your movements are only slowed slightly by your load.
h - a ring.
h - a egg.
i - a crude dagger.
h - a cursed crude dagger.
i - a ring.
the enemy picks up n crude daggers.
the enemy wields n crude daggers!
the enemy thrusts one of her crude daggers.
the bolt of cold whizzes by you!
the crude dagger welds itself to the goblin's hand!
h - a tripe ration.
you hit the enemy
your movements are now unencumbered.
h - a gem.
i - a enemy corpse.
the enemy picks up a gem.
h - a apple.
the enemy picks up a crude dagger.
h - a clear potion.
i - a skull cap.
you have a little trouble lifting h - a enemy corpse.
you have a little trouble lifting i - a food ration.
the crude dagger misses the enemy
```

F.4. MultiRoom-N{2,4}-Extreme

```
a crude dagger misses you.
a dart misses you.
g - a skull cap.
g - a enemy corpse.
g - a food ration.
h - a enemy corpse.
h - a lock pick.
h - a crude dagger.
whammm!!!
the enemy is killed!
the enemy just misses!
the enemy hits!
the enemy throws a crude dagger!
the enemy touches you!
the enemy jumps, nimbly evading your kick.
the enemy bites!
the enemy wields a crude dagger!
the enemy misses!
the enemy thrusts her crude dagger.
the little dog misses the enemy
the crude dagger misses the enemy
the kitten jumps, nimbly evading your kick.
the kitten bites the enemy
the kitten eats a enemy corpse.
the kitten misses the enemy
in what direction?
as you kick the door, it crashes open!
that hurts!
it burns up!
you are hit by a crude dagger.
you get zapped!
you see no door there.
you hit the enemy
you kill the enemy
you miss the enemy
you cannot escape from the enemy
you pull free from the enemy
you kick the enemy
you kick the kitten.
you kick at empty space.
you destroy the enemy
you strain a muscle.
you swap places with your kitten.
this door is locked.
what a strange direction!
do you want your possessions identified? ( n )
your leg feels better.
never mind.
really attack the little dog? ( n )
really attack the little dog? ( n ) n
really attack the kitten? ( n )
dumb move!
ouch!
g - a potion.
h - a potion.
h - a tripe ration.
the enemy escapes the dungeon!
the enemy misses the little dog.
the crude dagger hits the enemy
the dart misses the enemy
you are almost hit by a dart.
you swap places with your little dog.
do you want to see your attributes? ( n )
thump!
g - n darts.
h - n darts.
i - a enemy corpse.
j - a violet gem.
the enemy blocks your kick.
the enemy strikes at your displaced image and misses you!
the enemy throws a dart!
the little dog bites the enemy
the corpse misses the enemy
you are almost hit by a crude dagger.
you are hit by a dart.
you hear the rumble of distant thunder...
g - a crude dagger.
g - a yellowish gem.
g - n fortune cookies.
h - a wand.
h - n arrows.
the enemy picks up a crude dagger.
the enemy misses sirius.
the crude dagger slips as the enemy throws it!
you stop.
you swap places with hachi.
you swap places with sirius.
hachi misses the enemy
sirius bites the enemy
sirius misses the enemy
h - a skull cap.
sirius is in the way!
g - a scroll.
h - a faded pall.
h - a yellowish gem.
j - a enemy corpse.
the saddled pony drops a enemy corpse.
the saddled pony bites the enemy
g - a ring.
the saddled pony picks up a enemy corpse.
```

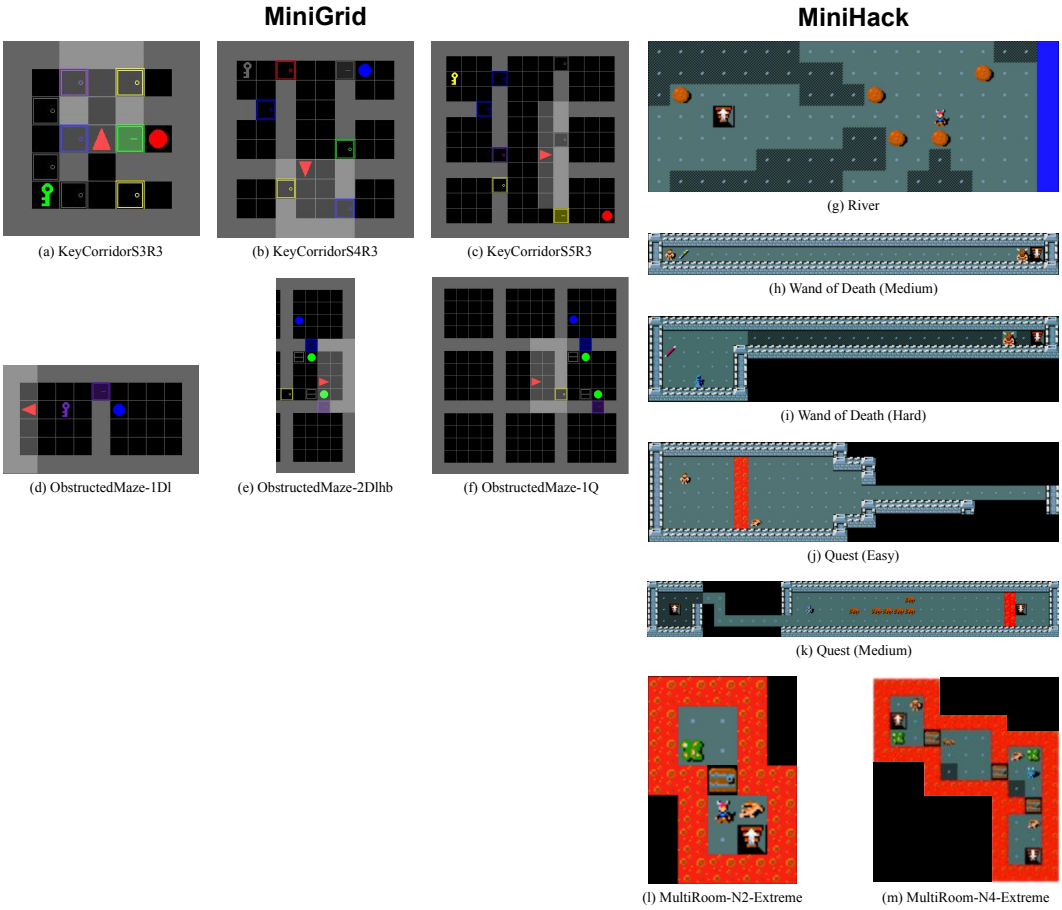


Figure S9: **Examples of all tasks evaluated in this work.** (a–f) MiniGrid tasks; (g–m) MiniHack tasks.