A Implementation Details

A.1 Environment

All environments used in this work were implemented using OpenAI Gym [4]. Below, we describe the MarlGrid Environments in more detail.

FindGoal The game map is a 15×15 grid world. On each episode reset, 1 goal tile and 25 obstacle tiles are randomly placed on the map. Then, 3 agents are randomly placed on the remaining empty space. Each agent can only observe a 7×7 partial view of the map centered on the agent. Each agent has 5 actions: up, right, down, left, stay. The maximum episode length allowed is 512 time steps.

RedBlueDoors The game map is a 10×10 grid world. On each episode reset, 1 blue door and 1 red door are placed at a random position on either the leftmost side or the rightmost side of the map; the doors must be on opposite sides. Then, 2 agents are randomly placed on the remaining empty space. Each agent can only observe a 3×3 partial view of the map centered on the agent. Each agent has 6 actions: up, right, down, left, stay, open door. The maximum episode length allowed is 2048 time steps.

A.2 Model

Image Encoder The Image Encoder is a convolutional neural network with 4 convolutional layers. Each layer has a kernel size of 3, stride of 2, padding of 1, and outputs 32 channels. ELU activation is applied to each convolutional layer. A 2D adaptive average pooling is applied over the output from convolutional layers. The final output has 32 channels, a height of 3, and a width of 3.

Communication Autoencoder The Communication Autoencoder takes as input the output from the Image Encoder. The encoder is a 3-layer MLP with hidden units [128, 64, 32] and ReLU activation. The decoder is a 3-layer MLP with hidden units [32, 64, 128] and ReLU activation. The output communication message is a 1D vector of length 10.

Message Encoder The Message Encoder first projects all input messages using an embedding layer of size 32, then concatenates and passes the message embeddings through a 3-layer MLP with hidden units [32, 64, 128] and ReLU activation. Dimension of the output message feature is 128.

Policy Network Each policy network is consisted of a GRU policy with hidden size 128, a linear layer mapping GRU outputs to policy logits for the environment action, and a linear layer mapping GRU outputs to the baseline value function.

B Training Details

B.1 Hyperparameters

For all experiments, we use the Adam optimizer [23] with a learning rate of 0.0001 and parameters $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$. We perform a sweep over the following values of these hyperparameter to select the combination that gives rise to best results: (0.01, 0.001, 0.0001, 0.00001) for learning rate, (0.9, 0.99, 0.995) for β_1, (0.995, 0.999) for β_2, and ($10^{-7}, 10^{-8}$) for ϵ.

B.2 Compute Resources

We ran all experiments on an internal cluster that consists of 4 NVIDIA GeForce RTX 2080 Ti GPUs.

14

methods	CIFAR	RedBlueDoors	FindGoal
	avg. $r \uparrow$	**avg.** $r \uparrow$	**avg.** $t \downarrow$
no-comm	0.082 ± 0.009	0.123 ± 0.096	169.0 ± 26.8
fc-comm	0.107 ± 0.021	0.382 ± 0.022	188.3 ± 33.6
latent-comm-1	0.129 ± 0.034	0.714 ± 0.073	160.0 ± 13.0
latent-comm-2	0.092 ± 0.012	0.739 ± 0.080	228.6 ± 21.3
full-latent-comm	0.346 ± 0.070	0.912 ± 0.046	111.1 ± 26.1
ae-comm (ours)	$\mathbf{0.348 \pm 0.041}$	$\mathbf{0.984 \pm 0.002}$	$\mathbf{103.5 \pm 20.2}$

Table 3: **Comparison with additional baselines.** We compute the average reward for `CIFAR Game` and `RedBlueDoors` environments, and average episode length for `FindGoal` environment. For each set of results, we report the mean and 95% confidence intervals evaluated on 100 episodes per seed over 10 random seeds.

C Additional Experiments

C.1 `fc-comm` and `latent-comm` Baselines

In addition to baselines presented in Section 5.2, we added two more baselines: `fc-comm` and `latent-comm`. Below, we describe them in more details, and report comparisons between these additional baselines and our method (`ae-comm`) on all three environments in Table 3.

For `fc-comm`, agents transmit discrete states that take the same form as the autoencoded messages in `ae-comm`. These discrete states come from the policy network without autoencoding. For fair comparison, a fully connected layer is added on top of the policy network states, such that the learned messages for `fc-comm` are discretized and have the same shape and range as messages in `ae-comm`.

For `latent-comm`, agents transmit latent states from before the policy head as communication vectors. For fair comparison, for `latent-comm` we again restrict agents to use the same fixed-size discrete communication channel (except for `full-latent-comm`, which we will explain in the next paragraph). The latent activation vectors in the policy nets are high-dimensional; reducing them to the size of the communication channel can be achieved in multiple ways. In `latent-comm-1`, we do so by reducing the size of the last hidden layer of the policy net so that it matches the size of the communication channel. In `latent-comm-2`, we instead use the original policy architecture and only transmit the first N features of the last hidden layer of the policy net, where N is the size of the communication channel. In all cases, we quantize the outputs identically and use a straight-through estimator [1] to differentiate through the quantization. Their performance is better than `fc-comm` baseline; since these latent states are continuously trained with the policy reward, we hypothesize that they contain more useful and relevant information.

As an "upper bound", we also compare all methods to `full-latent-comm`, a method that communicates the full, high-dimensional and continuous latent vector from the last hidden layer of the policy net. As shown in the table, the performance of `full-latent-comm` is close to that of `ae-comm`. Clearly, if agents can directly observe each other's full internal states, they can coordinate their behavior well. Our contribution is to show that similar performance can be achieved even when agents are only allowed to make low-dimensional and discrete utterances – a setting that more closely models communication in nature and may have greater practical utility (e.g., low-bandwidth communication channels between robots).

C.2 Training Emergent Communication via Policy Optimization

Making policy optimization work on top of autoencoding is a difficult yet essential problem to solve. We hypothesize that optimization is harder whenever the joint-exploration problem for learning speaker and listener policies is introduced. We do not have a solution yet, but our empirical results can confirm that the solution is not as simple as requiring more training iterations. We have run training to up to 1 million iterations and did not see an improvement in the result. We have also tried training the autoencoding component and RL component alternatingly, i.e. freezing one while

15
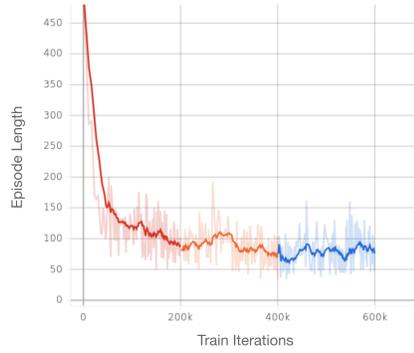
Figure 7: **Difficulty of training policy optimization:** We tried training the autoencoding component and RL component alternatingly, i.e. freezing one while training the other until loss stabilizes. The alternating training is visualized as different segments in this training graph, starting from the segment trained with autoencoding objective. Performance did not improve after extended training.

(a) RedBlueDoors ↑

(b) FindGoal ↓
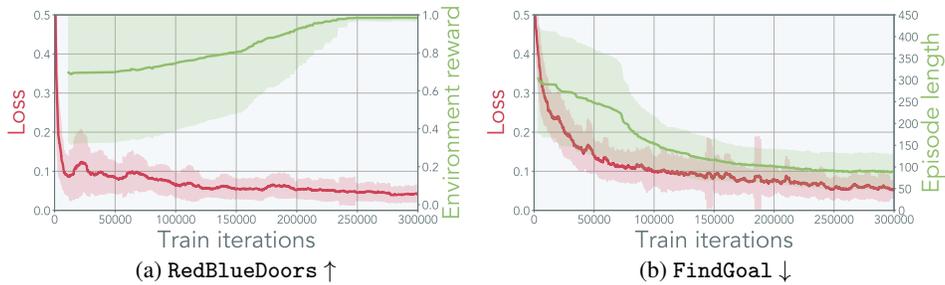
Figure 8: **Performance and representation:** Both agent performance and autoencoder loss improve over the course of learning.

training the other until loss stabilizes, yet performance does not improve whenever the RL objective is optimized. As an example, we include a failed training graph in Figure 7.

C.3   Autoencoder Loss

In Figure 8, we observe that agent task performance improves as representation learning task loss decreases. In particular, the agent task performance does not improve drastically until representation task loss is sufficiently low.

16