Grounded Language Learning with Uncertain Instructions

Stanford CS224N Custom Project

Ashish Rao
Department of Computer Science
Stanford University
aprao@stanford.edu

Kyle Hatch
Department of Computer Science
Stanford University
khatch@stanford.edu

Sylvia Yuan
Department of Computer Science
Stanford University
luyuan@stanford.edu

Abstract

This project explores the question of how one can train an Imitation Learning agent in an instruction-following environment when the instructions provided may be ambiguous. To answer this question, we design a system consisting of two modules: the Imitation Learning agent itself, and a classifier that can predict whether a provided instruction is ambiguous given the state of the environment. If the classifier classifies an instruction as ambiguous, the instruction must be clarified by a user. Otherwise, the Imitation Learning agent executes the instruction. Our system is evaluated on a variety of environments in the BabyAI platform that are modified to produce ambiguous instructions. We study and present results for two different classifier architectures – one based on a fine-tuned version of the GPT-2 model, and another based on an LSTM. We also show results comparing different ways to train the Imitation Learning agent. We find that using the classifier in conjunction with the Imitation Learning agent improves performance, with more significant gains on harder environments. We also see that the LSTM and GPT-2 models perform similarly in terms of accuracy, with the LSTM being marginally more accurate on the test set.

1 Key Information to include

- Mentor: Allan Zhou
- External Collaborators (if you have any): N/A
- Sharing project: No

2 Introduction

In recent years, the fields of Reinforcement Learning and Imitation Learning have seen an explosion of interest. Successes such as DeepMind's AlphaGo system to play the board fame Go and the OpenAI Five system to play Dota 2 have demonstrated the capability of Reinforcement Learning and Imitation Learning to be applied successfully to difficult problems. One exciting possibility is that of deploying RL/IL systems in the form of virtual assistants or home robots that can interact with and aid consumers. However, building systems that can have rich interactions with consumers entails building systems that are generally capable of understanding and responding to human language. A key component of this skill is recognizing and handling ambiguity – if a user provides an ambiguous

command to a virtual assistant or home robot, one would hope that the system would be able to recognize the ambiguity and ask for clarification if needed. This project studies precisely this setting, and investigates how one can train imitation learning agents to perform well in instruction-following environments when the instruction provided to the agent may be ambiguous.

3 Related Work

Much prior work has been done on RL/IL in instruction following environments. A large part of this work focuses on instruction following in 3D environments that simulate robot manipulation. For instance, Lynch et al [1] propose a method for Imitation Learning that utilizes a dataset of demonstrations, natural language descriptions, and goal states to solve instruction-following tasks in a 3D robot manipulation environment that requires the agent to open cupboards, manipulate objects, push sliding doors, and more. The BC-Z system by Jang et al [2] also performs imitation learning in a 3D robot manipulation setting and utilitizes a large dataset of human demos and interventions. Work has also been done on how language can be used internally in the agent as an abstraction to decompose complex, long-horizon tasks into sequences of lower level tasks in a hierarchical fashion. Jiang et al [3] investigate this setting and introduce the CLEVR robotics environment, which requires agents to arrange procedurally generated objects to satisfy some criteria.

Instead of instruction following for 3D robot manipulation environments, this project focuses on 2D gridworld instruction following environments. This is because solving 2D environments requires significantly less data and compute, and is thus a more practical choice than the 3D datasets mentioned above. In particular, we use modified environments from the BabyAI platform, which is a suite of 2D instruction following environments introduced by Chevalier-Boisvert et al [4]. The BabyAI environments are partially observable and feature varied instructions, and were originally introduced to study the sample efficiency of RL and IL algorithms in grounded language learning settings. We modify the BabyAI environments to produce ambiguous instructions with some probability in order to study IL algorithms in an ambiguous instruction following setting. The instructions produced by the BabyAI environments are actually generated automatically in accordance with a CFG. However, work by Marzoev et al [5] has studied how to effectively transfer BabyAI agents trained with synthetic instructions to natural language instructions generated by humans.

Despite the prior work on instruction following mentioned above, ambiguity in supplied instructions is a problem that has been less well studied. In this project, we develop a modular method to train Imitation Learning agents in ambiguous contexts and present results showing improvements in performance across several BabyAI environments modified to produce ambiguous instructions.

4 Approach

Our method to train IL agents on ambiguous instruction following environments is comprised of two distinct modules: a classifier to classify whether a given instruction is ambiguous that is also conditioned on the state, as well as the actual IL agent that executes tasks. If a given instruction is classified as unambiguous, this means that the user must clarify the instruction. Otherwise, the IL agent executes the task specified by the instruction. We develop and evaluate our system using 4 BabyAI environments: GoToLocal, PutNextLocal-S6N4, PickupLocal, and OpenDoorLoc. In our experiments, if the classifier classifies an instruction as ambiguous, the agent is hard-coded to query the environment for the fully specified instruction. We investigate two different architecture for the classifier: one based on a pretrained GPT-2 model from the HuggingFace library, and another being an LSTM model trained from scratch.

4.1 Ambiguous Instruction Generation

In order to generate instructions, we use a recursive method that operates on an internal tree-based representation of instructions used by the BabyAI environments. Each node corresponds to some kind of instruction, for example a ‘put’ instruction, or an ‘and’ instruction that links different instructions.

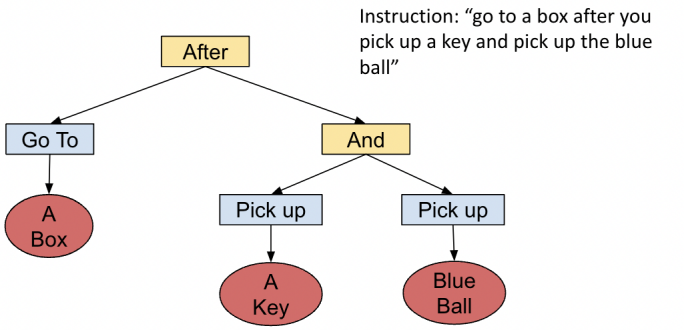


Figure 1: Internal Tree-Based Representation of Instructions

Instructions are made ambiguous by randomly making the subtrees of a node ambiguous, with each possible setting of which subtrees are ambiguous being equally likely (for instance, if an instruction has two subtrees, then there is equal probability for subtree A being ambiguous and B being unambiguous, A being ambiguous and B being unambiguous, and both being ambiguous). The leaves of the tree correspond to objects that the agent needs to interact with. When the algorithm reaches the leaves of the tree, we randomly drop attributes used to describe the object, for instance the object type (‘go to the ball’ $\rightarrow$ ‘go to the object’), the object location (‘the ball on your left’ $\rightarrow$ ‘the ball’), or the object color (‘the purple key’ $\rightarrow$ ‘the key’). In this way, we automatically generate ambiguous instructions for BabyAI environments. A key limitation of this method is that it is not guaranteed to provide ambiguous instructions. For instance, if the state contains only one box, and we turn the instruction “go to the yellow box” into “go to the box”, the modified instruction produced still fully specifies the task. In order to address this limitation, when generating labels for whether an instruction is ambiguous or not, we also perform a check in the environment for the number of objects that satisfy the object descriptions included in the environment. If the number of objects satisfying the object descriptions are more than one, then the instruction is marked as ambiguous, otherwise, it is marked as unambiguous.

4.2 GPT-2 Classifier Architecture

The classifier we use is a pre-trained GPT-2[6] model from the HuggingFace library fine tuned on the ambiguity classification task. The GPT-2 model is a decoder-only model, and given some input, the outputs of the model at the last token represent a probability distribution over next tokens. This probability distribution is produced by passing the input through several stacked decoder blocks. Each decoder block performs a linear transformation of the input, passes the input through a self-attention layer, and feeds the output through a feed-forward layer. Because GPT-2 uses last token to predict next tokens, the last embedding contains all information needed for prediction and can be used for classification[7].

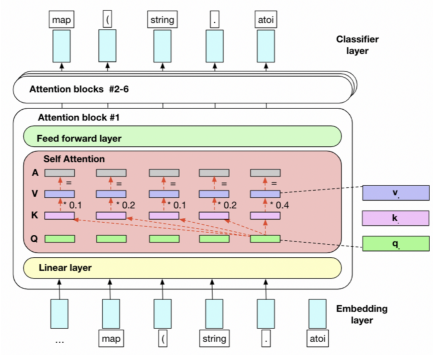


Figure 2: GPT-2 Architecture[8]

Since the label of whether or not an instruction is ambiguous also depends on the state of the environment, we cannot simply fine-tune the GPT-2 model according to the instructions alone. We instead concatenate the flattened current state from the environment and the instruction to form the input. This representation is then fed into GPT-2 in order to produce a prediction as to whether the given instruction is ambiguous or not.

4.3 LSTM Classifier Architecture

Long short-term memory[9][10] is a recurrent neural network variant that takes into account long term dependencies in sequential data. 5 shows LSTM architecture. Each LSTM unit contains forget gate, input gate, and output gate, which control information flow through the cell. An illustration of the LSTM archutecture is included in the Appendix.

An LSTM layer compute its cell states $\boldsymbol{c}$ and hidden states $\boldsymbol{h}$ from input $\boldsymbol{x}$ at timestep t with the following equations[11]:

$$
\begin{aligned}
\boldsymbol{f}^{(t)} &= \sigma\left(\boldsymbol{W}_f \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_f \boldsymbol{x}^{(t)} + \boldsymbol{b}_f\right) \\
\boldsymbol{i}^{(t)} &= \sigma\left(\boldsymbol{W}_i \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_i \boldsymbol{x}^{(t)} + \boldsymbol{b}_i\right) \\
\boldsymbol{o}^{(t)} &= \sigma\left(\boldsymbol{W}_o \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_o \boldsymbol{x}^{(t)} + \boldsymbol{b}_o\right) \\
\tilde{\boldsymbol{c}}^{(t)} &= \tanh\left(\boldsymbol{W}_c \boldsymbol{h}^{(t-1)} + \boldsymbol{U}_c \boldsymbol{x}^{(t)} + \boldsymbol{b}_c\right) \\
\boldsymbol{c}^{(t)} &= \boldsymbol{f}^{(t)} \circ \boldsymbol{c}^{(t-1)} + \boldsymbol{i}^{(t)} \circ \tilde{\boldsymbol{c}}^{(t)} \\
\boldsymbol{h}^{(t)} &= \boldsymbol{o}^{(t)} \circ \tanh \boldsymbol{c}^{(t)}
\end{aligned}
$$

4.4 Imitation Learning Agent Architecture

The Imitation Learning agent is built using the same architecture used for the BabyAI 1.1 baseline results [12]. The instruction is processed using a GRU, and the state using a CNN. The information from the instruction and state encoders are combined using FiLM (Feature-Wise Linear Modulation) layers. FiLM layers are a general method to condition the outputs of neural networks introduced by Perez et al [13]. In the architecture used, the FiLM layers perform an affine transformation of each of the filters of the CNN's output. The coefficients of the affine transformation are a learned function of the GRU-processed instruction.

Residual connections are also added after each FiLM layer. The output is then processed through a MaxPool and fed into an LSTM, which outputs predictions for the action to take and the value function for the input state. Illustrations of the FiLM layer and IL agent architecture are included in the appendix.

5 Experiments

5.1 Classifier Module

5.1.1 Data

Our dataset is 4000 instruction-state pairs, randomly selected from a large number of collected such pairs from different levels (OpenDoorLoc, GoToLocal, PickupLoc, PutNextLocalS6N4) in the BabyAI environment. We split the dataset into 2800 in training set, 800 in validation set, and 400 in test set. Around half of the instructions are turned ambiguous according to the method mentioned in previous section on ambiguous instructions. We label each pair by identifying all the object descriptions in the instruction, and searching for objects matching such description in the environment. If there are multiple objects matching one description, we label the instruction as ambiguous. We preprocess the data by concatenating tokenized instruction and flattened state and converting it to word indices vector, padded to maximum length of 156 as input to the models.

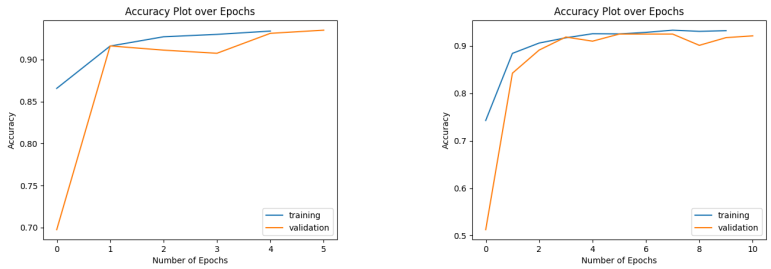
5.1.2 Evaluation Method

The evaluation metric is prediction accuracy. We seek to maximize accuracy in order to give more accurate classification results to the imitation learning agent in later experiments.

5.1.3 Experimental Details

We conducted experiments with the GPT-2 method and the LSTM method. With fine-tuning GPT-2, we used the pretrained 128M model, an AdamW optimizer with learning rate 0.00001, gradient clip norm to 1.0, and batch size of 16. We used binary cross entropy loss for calculating training and validation loss. Training took 4 hours. With LSTM, we used embedding dimension 256, droout layer with probability 0.2, an AdamW optimizer with learning rate 0.0001, gradient clip norm to 1.0, and batch size of 16. We used binary cross entropy loss for calculating training and validation loss. Training took 1 hour.

5.1.4 Results



(a) GPT-2 (b) LSTM

Figure 3: Accuracy Plots

	Train Accuracy	Val Accuracy	Test Accuracy
Finetuned GPT-2	0.9339	0.9350	0.9400
LSTM	0.9321	0.9250	0.9475

Table 1: Training, Validation, and Testing accuracy

Figure 3 shows the accuracy change over training epochs. GPT-2 converges slightly faster because of its pretrained nature. Both LSTM and Fine-tuned GPT-2 achieves similar accuracy of $\sim 94\%$ on the test set. Both models generalize well from the training set to validation and to the test set and achieve reasonable accuracy. We believe that both models should be sufficient to be used as classification module for the imitation learning agent.

5.2 Imitation Learning Agent

5.2.1 Data

We collected $1,000$ expert demonstrations on each of the four BabyAI environments we used: GoToLocal, PutNextLocal-S6N4, PickupLocal, and OpenDoorLoc (see section 4). We collected these demonstrations using the BabyAI environment BOT agent, which is a rules-based expert agent provided with the BabyAI codebase. Each demonstration consists of the instruction given by the environment for that episode and the trajectory executed by the BOT agent. In other words, for each of the four environments we collected a dataset of $1,000$ natural language instructions with $1,000$ corresponding expert trajectories. We then created three additional datasets per environment by altering the instructions into different conditions: a condition where 50% of the instructions were changed to be ambiguous (see section 4), a condition where 100% of the instructions were made ambiguous, and a condition where all of the instructions were replaced by a nonsense string (we used the string `ball ball ball ball ball ball ball ball ball ball`). In summary, for

each of the four environments, we obtained four datasets: a dataset where all of the insructions are unambiguous, a dataset where 50% of the instructions are ambiguous, a dataset where 100% of the instructions are ambiguous, and a dataset where all of the instructions are replaced with a nonsense string.

5.2.2 Experimental details

We trained 16 seperate immitation learning agents, one on each environment for each data condition described above. Each agent was trained for 50 million training iterations. We used the BabyAI imitation learning agent architecture described in section 4. We used all of the default imitation learning hyperparameters recommended by the BabyAI environment. Selected hyperparameters are shown below:

- **Loss:** Cross entropy loss with the targets being the expert actions.
- **Optimizer:** We used Adam optimization
- **Learning rate:** $3e-4$

5.2.3 Evaluation method

Every 500k training timesteps, we evaluted the current agent's policy was evaluated. Evaluation was carried out by executing the policy in the environment for 500 episodes and recording the average success rate. The success rate is defined as whether or not, during the course of an episode, the agent completes the task specified by the natural language instruction.

5.2.4 Results

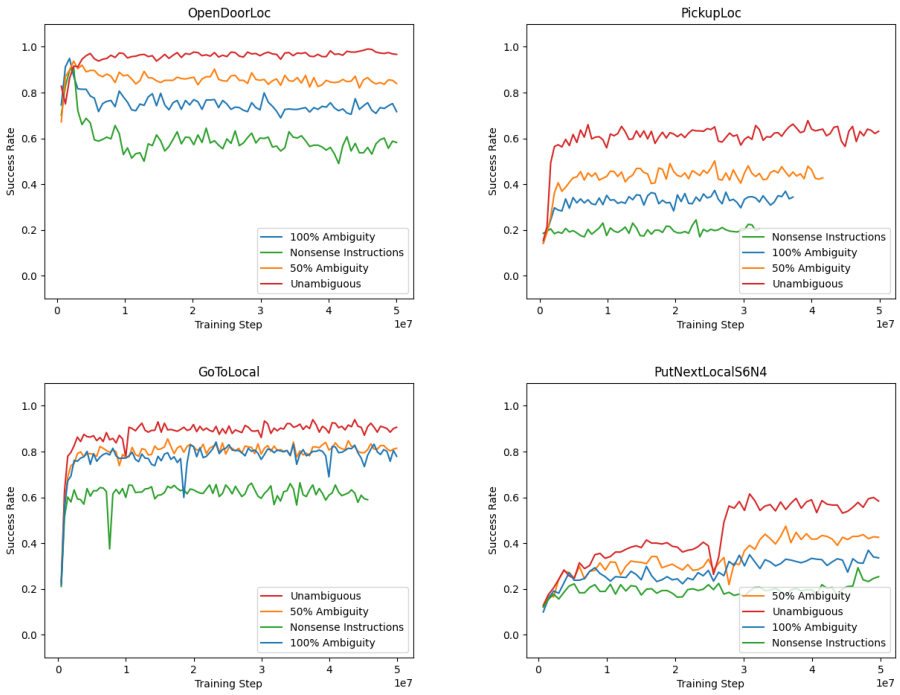


Figure 4: Imitation Learning Agent Training Success Rates. Across all environments, the IL agent that receives all unambiguous instructions (red curve) achieves the highest success rate. The agent that receives 50% ambiguous and 50% unambiguous instructions achieves the next highest success rate, the agent that receives 100% ambiguous instructions receives the second lowest success rate, and the agent that receives nonsense instructions receives the lowest success rate. An early stopping criterion was used: if the agent didn't improve its evaluation success rate in the past $1,000$ evaluations, then training was stopped.

The evaluation success rates of the different imitation learning agents is shown in figure 4. Across all four environments, the agent trained with completely unambiguous instructions achieves the highest success rate, the agent trained with 50% ambiguous and 50% unambiguous instructions achieved the second highest performance, the agent trained on 100% ambiguous instructions achieved the third highest success rate, and the agent trained on nonsense instructions achieved the lowest success rate. These results show that the imitation learning agents require having access to unambiguous instructions to be able to consistently solve the specified tasks. This demonstrates that the imitation learning agents actually learn to understand the different tasks specified by the natural language instructions and can solve them in the environments. Performing this ablation was necessary to confirm that the imitation learning agents would not be able to learn some trivial policy that could solve the tasks without using the environment instructions.

5.3 Integrated System

5.3.1 Evaluation method

With the classifier networks and the imitation learning agents trained separately, we now present the results of the integrated system. As described in section 4, each each timestep of an episode, a state-instruction pair is presented to our agent. If the classifier network identifies the instruction as ambiguous given the current state, then the agent queries the environment for a clarified instruction, and an unambiguous instruction is given to the agent. The imitation learning agent then chooses an action to perform given the current state and instruction. This action is passed to the environment, the environment is stepped, and the process repeats. The average success rate of the agent is recorded across many evaluation episodes.

We evaluate the performance of our integrated system across four BabyAI environments: GoToLocal, PutNextLocal-S6N4, PickupLocal, and OpenDoorLoc. Each of these environments is modified to output ambiguous instructions at a 50% rate and unambiguous instructions at a 50% rate. We compare the performance of our integrated system when using the lstm classifier network and the GPT-2 finetuned classifier network.

As a baseline, we compare against an agent that does not use a classifier network at all, meaning that it cannot query the environment to clarify ambiguous instructions. This provides a lower bound on the expected performance of our integrated system. Similarly, also compare against an agent that is evaluated on each of these environments without any ambiguous instructions (unambiguous instructions at a 100% rate). This is an oracle method that provides an upper bound on the expected performance of our integrated system.

5.3.2 Experimental details

We evaluate each agent for $1,000$ episodes. For evaluation, we use the classifier network weights that achieved the highest validation accuracy during training. We also use the imitation learning agent weights from the training epoch that achieved the best evaluation success rate during training. We use the agent weights that were trained on the datasets with 50% ambiguous instructions for all agents except for the oracle method. The oracle method uses the weights of the agent that was trained on the datasets with 100% unambiguous instructions, since it is evaluated on the completely unambiguous environments.

5.3.3 Results

	OpenDoorLoc	GoToLocal	PutNextLocal-S6N4	PickupLocal	Average
No Ambiguity (oracle)	98.3%	95.9%	67.2%	62.7%	81.0%
No Classifier (baseline)	99.7%	88.0%	51.4%	42.1%	70.3%
Integrated System with LSTM Classifier	99.5%	89.7%	53.2%	51.6%	73.5%
Integrated System with GPT-2 Classifier	99.5%	90.0%	55.0%	51.5%	74.0%

Table 2: Evaluation Success Rates of Integrated System. The evaluation success rates of the integrated system with the LSTM classifier and the GPT-2 Classifier are shown. The success rates of the baseline method and the oracle method are also shown.

The evaluation success rates of the integrated system using the LSTM classifier network and the GPT-2 classifier network (along with the baseline and oracle methods) are shown in table 2. Using the classifier networks leads to an increase in success rate in the GoToLocal, PutNextLocal-S6N4, and PickupLocal environments over the baseline, while there is not a significant different in performance in the OpenDoorLoc environments. The integrated system with either the LSTM or the GPT-2 classifier networks achieves a higher average performance than the baseline method does.

6 Analysis

Overall, the classifier models achieved reasonable accuracy and are able to predict whether an instruction is ambiguous based on text instruction and state from the environment. By printing out some falsely classified example from the classifier models, a significant portion of the errors occur when there are keywords dropped from the instruction, which means that we attempt to make the instruction ambiguous, but the instruction is not ambiguous according to the environment. The model sometimes fails to identify the non-ambiguity due to the environment. In addition, observing the results, we discover that pre-trained GPT-2 did not perform significantly better than LSTM trained from scratch. This observation is partially expected, since the flattened state expression is not natural language, and pretrained GPT-2 has no experience interpreting it.

The performance improvement given by using the classifier network in the integrated system is the largest in the environments in which the oracle method achieved the lowest success rate. The performance improvement is the lowest on the environments in which the oracle method achieved the highest success rate. This suggests that the benefits of our integrated system are most pronounced on difficult environments. Qualitative examination of the instructions given in the easy environments shows that they tend to be simple, one-step tasks involving a single object. For example, the OpenDoorLoc environment generates tasks such as "open the door in front of you" and "open a door on your right". In contrast, the more difficult environments generate instructions specifying performing multiple tasks with multiple objects. For example, the PutNextLocalS6N4 environment generates instructions such as "put the blue ball next to the yellow ball" and "put the green key next to the grey box". The longer, complicated instructions of the difficult environments make it so that any ambiguities in the instructions make solving the task very unlikely, while the agent may still be able to solve the relatively simple tasks of the easy environments even when the instructions are ambiguous. Therefore, querying for clarification about ambiguous instructions may be most beneficial on complex tasks that have long term dependencies, while on simple tasks asking for clarification does not seem to provide a significant boost in performance.

7 Conclusion

The ability to understand and execute natural language instructions is a crucial requirement for integrating artificially intelligent agents into real world applications. In real world scenarios, human users are likely to sometimes give ambiguous instructions to these agents. We model this problem setting in our modified BabyAI environments, and train two different neural network classifiers to detect when an instruction is ambiguous. We demonstrate that using this classifier to ask for clarification on ambiguous instructions significantly improves the performance of a language-conditioned imitation learning agent.

A limitation of our work is that we do not model the cost associated with asking for clarification. In real world applications, it is undesirable for an artificial intelligence agent to always be querying a human user, and therefore the cost of making such queries should be taken into consideration. This querying cost could naturally be modeled as a penalty term in an RL reward function. Training an RL agent to learn when it is worth it to ask for clarification about an instruction and when the cost of making such a request outweighs the benefits is a compelling direction for future research.

References

[1] Corey Lynch and Pierre Sermanet. Language conditioned imitation learning over unstructured data. *arXiv preprint arXiv:2005.07648*, 2020.

[2] Eric Jang, Alex Irpan, Mohi Khansari, Daniel Kappler, Frederik Ebert, Corey Lynch, Sergey Levine, and Chelsea Finn. Bc-z: Zero-shot task generalization with robotic imitation learning. In *Conference on Robot Learning*, pages 991–1002. PMLR, 2022.

[3] Yiding Jiang, Shixiang Shane Gu, Kevin P Murphy, and Chelsea Finn. Language as an abstraction for hierarchical deep reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[4] Maxime Chevalier-Boisvert, Dzmitry Bahdanau, Salem Lahlou, Lucas Willems, Chitwan Saharia, Thien Huu Nguyen, and Yoshua Bengio. Babyai: A platform to study the sample efficiency of grounded language learning. *arXiv preprint arXiv:1810.08272*, 2018.

[5] Alana Marzoev, Samuel Madden, M Frans Kaashoek, Michael Cafarella, and Jacob Andreas. Unnatural language processing: Bridging the gap between synthetic and natural language data. *arXiv preprint arXiv:2004.13645*, 2020.

[6] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[7] George Mihaila. Gpt-2 fine-tune classification. https://gmihaila.github.io/tutorial_notebooks/gpt2_finetune_classification/, 2021.

[8] Seohyun Kim, Jinman Zhao, Yuchi Tian, and Satish Chandra. Code prediction by feeding trees to transformers. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pages 150–162. IEEE, 2021.

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

[10] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

[11] Christopher Manning. Lecture 6: Simple and lstm recurrent neural networks. http://web.stanford.edu/class/cs224n/slides/cs224n-2022-lecture06-fancy-rnn.pdf, 2022.

[12] David Yu-Tung Hui, Maxime Chevalier-Boisvert, Dzmitry Bahdanau, and Yoshua Bengio. Babyai 1.1. *arXiv preprint arXiv:2007.12770*, 2020.

[13] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[14] Christopher Olah. Understanding lstm networks. 2015.

A Appendix

A.1 LSTM Classifier Architecture

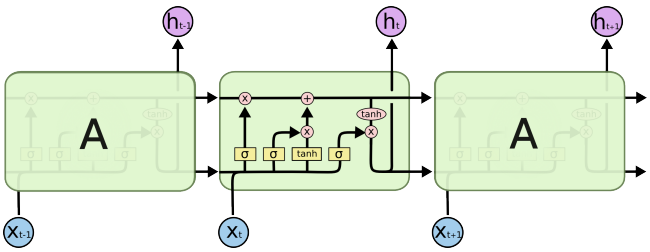


Figure 5: LSTM Architecture[14]

A.2 IL Agent Architecture Schematic

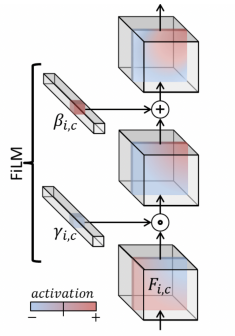


Figure 6: FiLM layer schematic

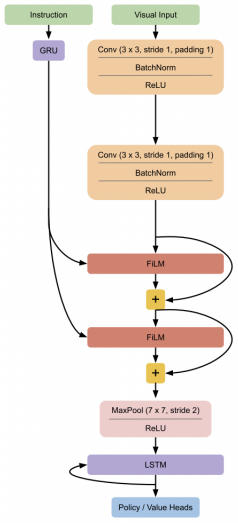


Figure 7: Imitation Learning agent schematic

A.3 RL Results

We also performed experiments to train RL agents on the BabyAI environments. We found that the RL agents learned a trivial policy of simply interacting with as many objects in the environment as possible to maximize their success rate. This led to identical performance for the RL agents regardless of the type of instruction they were supplied (unambiguous, ambiguous, or nonsensical). To remedy this problem, we attempted to introduce a penalty at each timestep to discourage the long episode lengths that result from the trivial policy learned, however this was insufficient to prevent the RL agents from learning a trivial policy.

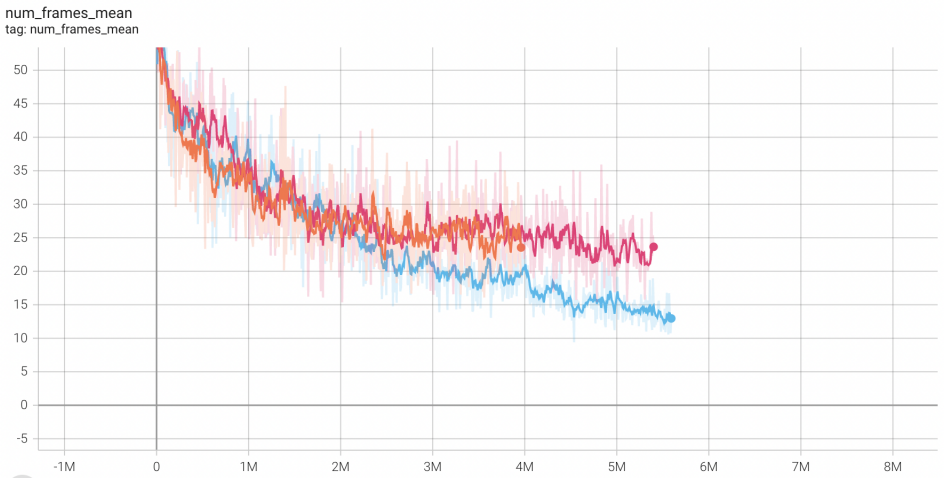


Figure 8: Identical episode lengths across different instruction types. Orange: ambiguous, blue: plain, magenta: nonsense.

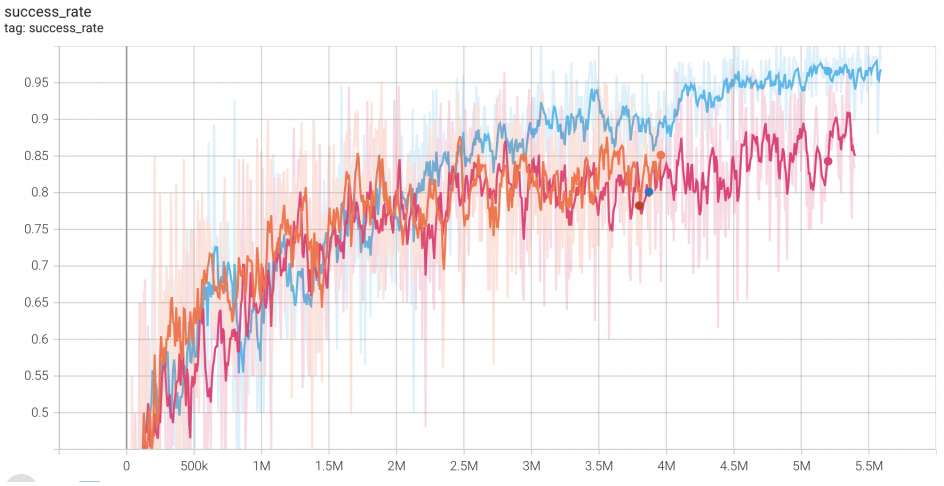


Figure 9: Identical success rates across different instruction types. Orange: ambiguous, blue: plain, magenta: nonsense.