Interactive Learning from Activity Description

Khanh Nguyen [1] **Dipendra Misra** [2] **Robert Schapire** [2] **Miro Dudík** [2] **Patrick Shafto** [3]

Abstract

We present a novel interactive learning protocol that enables training request-fulfilling agents by verbally describing their activities. Our protocol gives rise to a new family of interactive learning algorithms that offer complementary advantages against traditional algorithms like imitation learning (IL) and reinforcement learning (RL). We develop an algorithm that practically implements this protocol and employ it to train agents in two challenging request-fulfilling problems using purely language-description feedback. Empirical results demonstrate the strengths of our algorithm: compared to RL baselines, it is more sample-efficient; compared to IL baselines, it achieves competitive success rates while not requiring feedback providers to have agent-specific expertise. We also provide theoretical guarantees of the algorithm under certain assumptions on the teacher and the environment.

1. Introduction

The goal of a *request-fulfilling* agent is to map a given language request in a situated environment to an execution that accomplishes the intent of the request (Winograd, 1972; Chen & Mooney, 2011; Tellex et al., 2012; Artzi et al., 2013; Misra et al., 2017; Anderson et al., 2018; Chen et al., 2019; Nguyen et al., 2019; Nguyen & Daumé III, 2019; Gaddy & Klein, 2019). Developing request-fulfilling agents is an important step towards creating autonomous assistants that communicate with humans naturally. Request-fulfilling agents have been typically trained using *non-verbal* interactive learning protocols such as imitation learning (IL) which assumes labeled executions as feedback (Mei et al., 2016; Anderson et al., 2018), or reinforcement learning (RL) which uses scalar rewards as feedback (Chaplot et al., 2018; Hermann et al., 2017). We introduce a new interactive learning protocol for training these agents called ILIAD: **I**nteractive **L**earning from **A**ctivity **D**escription, where feedback is limited to *language descriptions of executions*.

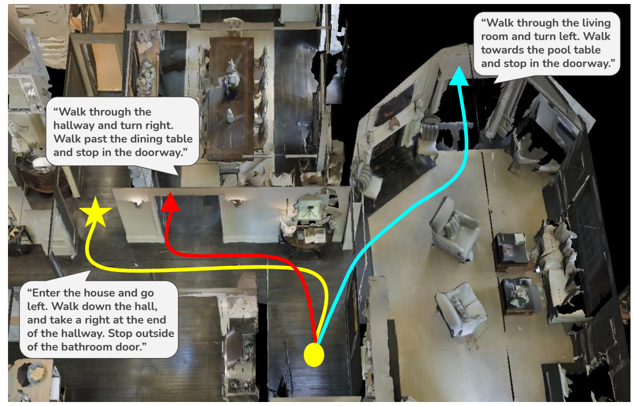


Figure 1. A real example of training an agent to fulfill a navigation request in 3D environments (Anderson et al., 2018) using ADEL, our implementation of the ILIAD protocol. The agent receives a request *"Enter the house..."* which implies the path. Initially, it does not understand language and thus wanders far from the goal. Its execution (the path) is described as *"Walk through the living room..."*. To ground the description language, the agent learns to generate the path conditioned on the description. After a number of interactions, its execution () is closer to the optimal path. As this process iterates, the agent gradually improves its understanding of the description language and thus also executes requests more precisely.

Figure 1 illustrates an example of training an agent to fulfill a navigation request using the ILIAD protocol. Learning proceeds in episodes of interaction between a learning agent and a teacher. In each episode, the agent is presented with a language request and takes a sequence of actions in the environment to execute the request. The agent is assumed to have no prior language understanding capability, and therefore, cannot fulfill the request in the beginning. The teacher enables language understanding by providing the agent with a *language description of the agent's execution*. We assume that the descriptions are specified in the same language that is used to specify the requests. Hence, by grounding the descriptions to the corresponding executions, the agent can acquire knowledge about the description language and thus can also improve its request-fulfilling capability. Crucially,

[1]Department of Computer Science, University of Maryland, Maryland, USA [2]Microsoft Research, New York, USA [3]Rutgers University, New Jersey, USA. Correspondence to: Khanh Nguyen <kxnguyen@umd.edu>.

Table 1. Trade-offs between the learning effort of the agent and the teacher in learning protocols. Each protocol employs a different medium for the teacher to convey feedback. If a medium is not natural to the teacher (e.g. IL-style demonstration), it must learn to encode feedback intent using that medium (*teacher communication-learning effort*). Similarly, if a medium is not natural to the agent (e.g. human language), it needs to learn to interpret feedback (*agent communication-learning effort*). The agent also learns tasks from information decoded from feedback (*agent task-learning effort*). The qualitative claims on the "agent learning effort" column summarize our empirical findings on the learning efficiency (measured by sample complexity) of these protocols.

		Learning effort	
Protocol	Feedback medium	Teacher (communication learning)	Agent (comm. & task learning)
IL	Demonstration	Highest	Lowest
RL	Scalar reward	None	Highest
ILIAD	Language description	None	Medium

the agent receives *no other* feedback such as ground-truth demonstration (Mei et al., 2016), scalar reward (Hermann et al., 2017), or constraint (Miryoosefi et al., 2019). At test time, the teacher is not present and the agent must execute requests autonomously.

The ILIAD protocol leaves two open problems for the agent: (a) how to generate executions that elicit useful descriptions from the teacher and (b) how to effectively learn the description language. We develop an algorithm named ADEL: **A**ctivity-**D**escription **E**xplorative **L**earner that offers practical solutions to these problems. For (a), we devise a semi-supervised sampling scheme that efficiently explores the execution space. For (b), we employ behavior cloning (Pomerleau, 1991) to ground descriptions to executions. We theoretically prove convergence for a variant of ADEL in the contextual bandit setting (Langford & Zhang, 2008b).

Our paper does *not* argue for the primacy of one learning protocol over the others. In fact, an important point we raise is that there are multiple, possibly competing metrics for comparing learning protocols. We focus on the trade-off between the learning effort of the agent and the teacher in each protocol (Table 1). In all protocols, the agent and the teacher establish a communication channel that allows the teacher to encode feedback and send it to the agent, who learns tasks based on information decoded from feedback. At one extreme, IL places the burden of establishing the communication channel entirely on the teacher. To provide a demonstration, the teacher in IL must learn to control the agent to accomplish tasks by specifying actions that lie in the agent's action space.[1] To compensate for this effort, the agent usually learns very efficiently with IL because it does not have to learn to interpret feedback, and the feedback directly specifies desired behavior. At another extreme, we have RL and ILIAD, where the teacher provides feedback via agent-agnostic media (reward and language, respectively). RL eliminates the agent communication-learning effort by hard-coding the semantics of scalar rewards into the learning algorithm.[2] But the trade-off of using such limited feedback is that the effort required by the agent to learn the task increases. State-of-the-art RL algorithms are notorious for their high sample complexity, making them expensive to use outside simulators (Hermann et al., 2017; Chaplot et al., 2018; Chevalier-Boisvert et al., 2019). By employing a natural and highly expressive medium like natural language, ILIAD offers a compromise between RL and IL: it is more sample-efficient than RL while not requiring the teacher to master the agent's control interface. Overall, no protocol is superior in all metrics and the choice of protocol depends on users' preferences.

We empirically evaluate ADEL against IL and RL baselines on two tasks: vision-language navigation (Anderson et al., 2018), and word-modification via regular expressions (Andreas et al., 2018). Our results show that ADEL significantly outperforms RL baselines in terms of both learning efficiency and effectiveness. On the other hand, ADEL's success rate is competitive with those of the IL baselines on the navigation task and is lower by 4% on the word modification task. It takes approximately 5-8 times more training episodes than the IL baselines to reach comparable success rates, which is quite respectable considering that the algorithm has to search in an exponentially large space for the ground-truth executions whereas the IL baselines are *given* these executions. Therefore, ADEL can be a preferred algorithm whenever annotating ground-truth executions is not feasible or is substantially more expensive than describing executions. For example, in the word-modification task, ADEL teaches the agent without requiring a teacher with knowledge about regular expressions, who can be costly to recruit in practice. We believe the capability of non-experts to provide feedback will make ADEL and more generally the ILIAD protocol a strong contender in many scenarios.

2. ILIAD: Interactive Learning from Activity Descriptions

Environment. We borrow our terminology from the reinforcement learning (RL) literature (Sutton & Barto, 2018). We consider an agent acting in an environment with state

[1]Third-person or observational IL (Stadie et al., 2017; Sun et al., 2019) allows the teacher to demonstrate tasks with their action space. However, this framework is *non-interactive* because the agent imitates pre-collected demonstrations and does not interact with a teacher. We consider interactive IL (Ross et al., 2011), which is shown to be more effective than non-interactive counterparts.

[2]By design, RL algorithms understand that higher reward value implies better performance.

Algorithm 1 ILIAD protocol. Details of line 4 and line 6 are left to specific implementations.

1: Initialize agent policy $\pi_\theta : \mathcal{S} \times \mathcal{D} \to \Delta(\mathcal{A})$
2: **for** $n = 1, 2, \cdots, N$ **do**
3: World samples $q = (R, d^\star, s_1) \sim \mathbb{P}^\star(\cdot)$
4: Agent generates execution $\hat{e}$ given π_θ, $d^\star$, and s_1
5: Teacher generates description $\hat{d} \sim \mathbb{P}_T(\cdot \mid \hat{e})$
6: Agent uses $\left(d^\star, \hat{e}, \hat{d}\right)$ to update π_θ
return π_θ

space $\mathcal{S}$, action space $\mathcal{A}$, and transition function $T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$, where $\Delta(\mathcal{S})$ denotes the space of all probability distributions over $\mathcal{S}$. Let $\mathcal{R} = \{R : \mathcal{S} \times \mathcal{A} \to [0, 1]\}$ be a set of reward functions. A *task* in the environment is defined by a fixed choice of reward function $R \in \mathcal{R}$. The agent does not have access to a task's reward function. Instead, the task is specified to the agent via a *language request* $d^\star \in \mathcal{D}$, where $\mathcal{D}$ is the set of all nonempty strings generated from a finite vocabulary. For example, in robot navigation, a request "*go to the kitchen*" specifies a task given by a reward function that is maximized when the robot is in the kitchen.

Execution Episode. At the beginning of an episode, a task $q = (R, d^\star, s_1)$ of reward function R, request $d^\star$, and start state s_1 is sampled from a task distribution $\mathbb{P}^\star(q)$. The agent starts in s_1 and is presented with $d^\star$ but *does not* observe R or any rewards generated by it. The agent maintains a *request-conditioned policy* $\pi_\theta : \mathcal{S} \times \mathcal{D} \to \Delta(\mathcal{A})$ with parameters θ, which takes in a state $s \in \mathcal{S}$ and a request $d \in \mathcal{D}$, and outputs a probability distribution over $\mathcal{A}$. Using this policy, it can generate an *execution* $\hat{e} = (s_1, \hat{a}_1, s_2, \cdots, s_H, \hat{a}_H)$, where H is the task horizon (the time limit), $\hat{a}_i \sim \pi_\theta(\cdot \mid s_i, d^\star)$ and $s_{i+1} \sim T(\cdot \mid s_i, \hat{a}_i)$ for every i. Throughout the paper, we will use the notation $e \sim \mathbb{P}_\pi(\cdot \mid s_1, d)$ to denote sampling an execution e by following policy π given a start state s_1 and a request d. The objective of the agent is to find a policy π with maximum value, where we define the policy value $V(\pi)$ as:

$$V(\pi) = \mathbb{E}_{q \sim \mathbb{P}^\star(\cdot), \hat{e} \sim \mathbb{P}_\pi(\cdot \mid s_1, d^\star)} \left[\sum_{i=1}^{H} R(s_i, \hat{a}_i) \right] \quad (1)$$

ILIAD protocol. Alg 1 describes the ILIAD protocol for training a request-fulfilling agent. It consists of a series of N training episodes, where each episode starts with sampling a task $q = (R, d^\star, s_1)$ from $\mathbb{P}^\star$. The agent starts in s_1, receives $d^\star$, and generates an execution $\hat{e}$ (line 4). The feedback mechanism in ILIAD is provided by a *teacher* that can describe executions in language. The teacher is modeled by a fixed distribution $\mathbb{P}_T : (\mathcal{S} \times \mathcal{A})^H \to \Delta(\mathcal{D})$, where $(\mathcal{S} \times \mathcal{A})^H$ is the space over H-step executions. After completing its execution $\hat{e}$, the agent sends it to the teacher and receives a *language description* of $\hat{e}$, which is a sample $\hat{d} \sim \mathbb{P}_T(\cdot \mid \hat{e})$ (line 5). Finally, the agent uses the triplet $\left(d^\star, \hat{e}, \hat{d}\right)$ to update its policy for the next round (line 6). Crucially, the agent *never* receives any other feedback, including rewards, demonstrations, constraints, or direct knowledge of the *latent* reward function. Any algorithm implementing ADEL protocol has to decide how to generate executions (line 4) and how to update the agent policy (line 6). The protocol does not provide any constraints for these decisions.

Consistency of the teacher. In order for the agent to learn to execute requests by grounding the description language, we must require that the description language is similar to the request language. Formally, we define the ground-truth joint distribution over tasks and executions as follows

$$\mathbb{P}^\star(e, R, s_1, d) = \mathbb{P}_{\pi^\star}(e \mid s_1, d)\, \mathbb{P}^\star(R, d, s_1) \quad (2)$$

where $\pi^\star$ is the optimal policy that maximizes Eq 1. From this joint distribution, we derive the ground-truth conditional distribution over requests given execution $\mathbb{P}^\star(d \mid e)$. This distribution specifies the probability that a request d can serve as a valid description of an execution e.

We expect that if $\mathbb{P}_T(d \mid e)$ is close to $\mathbb{P}^\star(d \mid e)$ then learning to understand descriptions will help with request-fulfilling. In that case, the agent can treat a description of an execution as a request that is fulfilled by that execution. Therefore, the description-execution pairs $\left(\hat{d}, \hat{e}\right)$ can be used as supervised-learning examples for the request fulfilling problem.

The learning process can also be sped up if the agent is able to exploit the compositionality of language. For example, if a request is "*turn right, walk to the kitchen*" and the agent's execution is described as "*turn left, walk to bedroom*", the agent may not have successfully fulfilled the task but it can learn what "*turn*" and "*walk to*" mean through the description. Later, it may learn to recognize "*kitchen*" through a description "*go to the kitchen*" and compose that knowledge with its understanding of "*walk to*" to better execute "*walk to the kitchen*".

3. ADEL: Learning from Activity Describers via Semi-Supervised Exploration

We frame the ILIAD problem as follows: *given that we can effectively draw samples from the marginal* $\mathbb{P}^\star(s_1, d)$ *and a consistent teacher* $\mathbb{P}_T(d \mid e)$*, how do we learn a policy* π_θ *such that* $\mathbb{P}_{\pi_\theta}(e \mid s_1, d)$ *is close to* $\mathbb{P}^\star(e \mid s_1, d)$*?* Here, $\mathbb{P}^\star(e \mid s_1, d) = \mathbb{P}_{\pi^\star}(e \mid s_1, d)$ is the ground-truth request-fulfilling distribution obtained from the joint distribution

Algorithm 2 Simple algorithm for estimating $\mathbb{P}^\star(e \mid s_1, d)$ with access to the true marginal $\mathbb{P}^\star(e \mid s_1)$ and teacher $\mathbb{P}_T(d \mid e)$.

```
B = ∅
for i = 1, 2, ⋯ , N do
    World samples task q = (R, d⋆, s_1) ∼ P⋆(·)
    Sample (ê, d̂) as follows: ê ∼ P⋆(· | s_1), d̂ ∼ P_T(· | ê)
    B ← B ∪ {(ê, d̂)}
Train a policy π_θ(a | s, d) via maximum log-likelihood:
    max_θ Σ_{(ê,d̂)∈B} Σ_{(s,a_s)∈ê} log π_θ(a_s | s, d̂)
    where a_s is the action taken by the agent in state s
return π_θ
```

defined in Eq 2. We have

$$\begin{aligned}\mathbb{P}^\star(e \mid s_1, d) &\propto \mathbb{P}^\star(e \mid s_1)\mathbb{P}^\star(d \mid e)\\ &\approx \mathbb{P}^\star(e \mid s_1)\mathbb{P}_T(d \mid e)\end{aligned} \tag{3}$$

As seen from the equation, the only missing piece required for estimating $\mathbb{P}^\star(e \mid s_1, d)$ is the (state-dependent) marginal $\mathbb{P}^\star(e \mid s_1)$. Alg 2 presents a simple method for learning an agent policy if we have access to this marginal. It is easy to show that the pairs $\left(\hat{e}, \hat{d}\right)$ in the algorithm are approximately drawn from the joint distribution $\mathbb{P}^\star(e, d)$ and thus can be directly used to estimate the conditional $\mathbb{P}^\star(e \mid s_1, d)$.

Unfortunately, $\mathbb{P}^\star(e \mid s_1)$ is usually unknown in our setting. We present our main algorithm ADEL (Alg 3) which simultaneously estimates $\mathbb{P}^\star(e \mid s_1)$ and $\mathbb{P}^\star(e \mid s_1, d)$ through interactions with the teacher. In this algorithm, we assume access to an *approximate marginal* $\mathbb{P}_{\pi_\omega}(e \mid s_1)$ defined by an *explorative policy* $\pi_\omega(a \mid s)$. This policy can be learned from a dataset of unlabeled executions or be defined as a program that synthesizes executions. In many applications, reasonable unlabeled executions can be cheaply constructed using knowledge about the structure of the execution. For example, in robot navigation, valid executions are collision-free and non-looping; in semantic parsing, predicted parses should follow the syntax of the semantic language.

After constructing the approximate marginal $\mathbb{P}_{\pi_\omega}(e \mid s_1)$, we could substitute it for the true marginal in Alg 2. However, using a fixed approximation of the marginal may lead to sample inefficiency when there is a mismatch between the approximate marginal and the true marginal. For example, in the robot navigation example, if most human requests specify the kitchen as the destination, the agent should focus on generating executions that end in the kitchen to obtain descriptions that are similar to those requests. If instead, a uniform approximate marginal is used to generate executions, the agent obtains a lot of irrelevant descriptions.

ADEL minimizes potential marginal mismatch by iteratively using the estimate of the marginal $\mathbb{P}^\star(e \mid s_1)$ to improve

Algorithm 3 ADEL: our implementation of the ILIAD protocol.

```
1: Input: teacher model P_T(d | e), approximate marginal
   P_{π_ω}(e | s_1), mixing rate λ ∈ [0, 1], annealing rate β ∈ (0, 1)
2: Initialize π_θ : S × D → Δ(A) and B = ∅
3: for n = 1, 2, ⋯ , N do
4:     World samples task q = (R, d⋆, s_1) ∼ P⋆(·)
5:     Agent generates ê ∼ P̃(· | s_1, d⋆) (see Eq 4)
6:     Teacher generates description d̂ ∼ P_T(· | ê)
7:     B ← B ∪ (ê, d̂)
8:     Update agent policy:
           θ ← max_{θ'} Σ_{(ê,d̂)∈B} Σ_{(s,a_s)∈ê} log π_{θ'}(a_s | s, d̂)
       where a_s is the action taken by the agent in state s
9:     Anneal mixing rate: λ ← λ · β
   return π_θ
```

the estimate of the conditional $\mathbb{P}^\star(e \mid s_1, d)$ and vice versa. Initially, we set $\mathbb{P}_{\pi_\omega}(e \mid s_1)$ as the marginal over executions. In each episode, we *mix* this distribution with $\mathbb{P}_{\pi_\theta}(e \mid s_1, d)$, the current estimate of the conditional, to obtain an improved estimate of the marginal (line 5). Formally, given a start state s_1 and a request $d^\star$, we sample an execution $\hat{e}$ from the following distribution:

$$\tilde{\mathbb{P}}(\cdot \mid s_1, d^\star) \triangleq \lambda \mathbb{P}_{\pi_\omega}(\cdot \mid s_1) + (1-\lambda)\mathbb{P}_{\pi_\theta}(\cdot \mid s_1, d^\star) \tag{4}$$

where $\lambda \in [0, 1]$ is a mixing rate that is annealed to zero over the course of training. Each component of the mixture in Eq 4 is essential in different learning stages. Mixing with $\mathbb{P}_{\pi_\omega}$ accelerates convergence at the early stage of learning. Later, when π_θ improves, $\mathbb{P}_{\pi_\theta}$ skews $\tilde{\mathbb{P}}$ towards executions whose descriptions are closer to the human requests, closing the gap with $\mathbb{P}^\star(e \mid s_1)$. In line 6-8, similar to Alg 2, we leverage the (improved) marginal estimate and the teacher to draw samples $\left(\hat{e}, \hat{d}\right)$ and use them to re-estimate $\mathbb{P}_{\pi_\theta}$.

Theoretical Analysis. We analyze an epoch-based variant of ADEL and show that under certain assumptions, it converges to a near-optimal policy. In this variant, we run the algorithm in epochs, where the agent policy is only updated at the end of an epoch. In each epoch, we collect a fresh batch of examples $\left\{\left(\hat{e}, \hat{d}\right)\right\}$ as in ADEL (line 4-7), and use them to perform a batch update (line 8). We provide a sketch of our theoretical results here and defer the full details to Appendix A.

We consider the case of $H = 1$ where an execution $e = (s_1, a)$ consists of the start state s_1 and a single action a taken by the agent. This setting while restricted captures the non-trivial class of contextual bandit problems (Langford & Zhang, 2008b). Sequential decision-making problems where the agent makes decisions solely based on the start state can be reduced to this setting by treating a sequence of decisions as a single action (Kreutzer et al., 2017; Nguyen et al., 2017a). We focus on the convergence of the iterations of epochs, and assume that the maximum likelihood estimation problem in each epoch can be solved optimally. We also ablate the teacher learning difficulty by assuming access to a fully consistent teacher, i.e., $\mathbb{P}_T(d \mid e) = \mathbb{P}^\star(d \mid e)$.

We make two crucial assumptions. Firstly, we make a standard realizability assumption to ensure that our policy class is expressive enough to accommodate the optimal solution of the maximum likelihood estimation. Secondly, we assume that for every start state s_1, the teacher matrix $\mathbb{P}^\star(d \mid e_{s_1})$ over descriptions and executions e_{s_1} starting with s_1, has a non-zero minimum singular value $\sigma_{min}(s_1)$. Intuitively, this assumption implies that descriptions are rich enough to help in deciphering actions. Under these assumptions, we prove the following result:

Theorem 1 (Main Result). *Let $\mathbb{P}_n(e \mid s_1)$ be the marginal distribution in the n^{th} iteration. Then for any $t \in \mathbb{N}$ and any start state s_1 we have:*

$$\|\mathbb{P}^\star(e \mid s_1) - \frac{1}{t}\sum_{n=1}^{t} \mathbb{P}_n(e \mid s_1)\|_2 \leq \frac{1}{\sigma_{min}(s_1)}\sqrt{\frac{2\ln|\mathcal{A}|}{t}}.$$

Theorem 1 shows that running average of the marginal distribution converges to the true marginal distribution. As argued before, access to the true marginal can be easily used to learn a near-optimal policy. For brevity, we defer proof and other details to Appendix A. Our results show that under certain conditions, we can expect convergence to the optimal policy. We leave the question of sample complexity and addressing more general settings for future work.

4. Related Work

Frameworks for learning from language-based communication have been previously proposed. Common approaches include: reducing the learning problem to reinforcement learning (Goldwasser & Roth, 2014; MacGlashan et al., 2015; Ling & Fidler, 2017; Goyal et al., 2019; Fu et al., 2019; Sumers et al., 2020), grounding language to demonstration (Chen & Mooney, 2011; Misra et al., 2014; Bisk et al., 2016; Liu et al., 2016; Wang et al., 2016; Li et al., 2017; 2020a;b), or devising EM-based algorithms to parse language into logical forms (Matuszek et al., 2012; Labutov et al., 2018). The first approach may discard useful learning signals from language feedback and inherits the limitations of RL algorithms. The second requires extra effort from the teacher to provide demonstrations. The third approach has to bootstrap the language parser with labeled executions. ADEL enables learning from a specific type of language feedback (language description) without reducing it to reward, requiring demonstrations, or assuming access to labeled executions. Recently, several papers have proposed using language description feedback for speeding up reinforcement learning (Jiang et al., 2019) or aiding exploration of new skills (Colas et al., 2020). We provide a detailed comparison with these papers in Appendix B.

Another related line of research is work on the rational speech act (RSA) model (Grice, 1975; Golland et al., 2010; Monroe & Potts, 2015; Goodman & Frank, 2016; Andreas & Klein, 2016; Fried et al., 2018), which is also concerned about transferring information via language. However, it is important to point out that RSA is a mental *reasoning* model whereas ILIAD is an *interactive* protocol. In RSA, a speaker (or a listener) constructs a pragmatic message-encoding (or decoding) scheme by building an internal model of a listener (or a speaker). Importantly, during that process, one agent *never* interacts with the other. In contrast, the ILIAD agent learns through interaction with a teacher. In addition, RSA focuses on encoding (or decoding) a single message while ILIAD defines a process consisting of multiple rounds of message exchanging.

Finally, our work also fundamentally differs from work on (RL-based) cooperative emergent language (Lazaridou et al., 2017; Havrylov & Titov, 2017; Das et al., 2017; Evtimova et al., 2018; Kottur et al., 2017) in that we assume the teacher speaks a fixed, well-formed language, whereas in that work the teacher begins with no language capability and *evolves* a language over the course of learning.

5. Experimental Setup

In this section, we present a general method for simulating an execution-describing teacher using a pre-collected dataset (§5.1). Then we describe setups of the two problems we conduct experiments on: vision-language navigation (§5.2) and word modification (§5.3). Detail about the model architecture, training hyperparameters, and how the teacher is simulated in each problem is in the Appendix.

5.1. Simulating Teachers

ILIAD assumes access to a teacher $\mathbb{P}_T(d \mid e)$ that can reliably describe agent executions. For our experimental purposes, employing human teachers is expensive and irreproducible, thus we simulate them using pre-collected datasets. We assume availability of a dataset $\mathcal{B}_{\text{sim}} = \{(\mathcal{D}_n, e_n)\}_{i=1}^{N}$, where $\mathcal{D}_n = \left\{d_n^{(j)}\right\}_{j=1}^{M}$ contains M human-generated requests that are fulfilled by execution e_n. The agent does *not* have direct access to this dataset; it only observes the data points in the dataset by communicating with the simulated teacher following the ILIAD protocol.

Each ILIAD episode requires providing a request $d^\star$ at the beginning and a description $\hat{d}$ of an agent execution $\hat{e}$. The request $d^\star$ is chosen by first uniformly randomly selecting an example $(\mathcal{D}_n, e_n)$ from $\mathcal{B}_{\text{sim}}$, and then uniformly sampling a request $d^\star$ from $\mathcal{D}_n$. The description $\hat{d}$ is generated as follows. We first use all the pairs $\left(d_n^{(j)}, e_n\right)$ in $\mathcal{B}_{\text{sim}}$ to train an RNN-based conditional language model $\tilde{\mathbb{P}}_T(d \mid e)$ via standard maximum log-likelihood. We can then generate a description of an execution by greedily decoding[3] this model conditioned on the execution. However, given limited training data, this model may not generate sufficiently high-quality descriptions. Hence, we also make use of the human-generated requests to provide higher-quality descriptions. Let $\texttt{perf}\,(\hat{e}, e_n)$[4] be a metric that evaluates an agent execution $\hat{e}$ against a ground-truth e_n (higher is better). If $\texttt{perf}\,(\hat{e}, e_n)$ is less than a pre-defined threshold τ, we generate a description by greedily decoding the model $\tilde{\mathbb{P}}_T(\cdot \mid \hat{e})$; otherwise, we uniformly sample a request from $\mathcal{D}_n$ and return it as the description. Implementation details of `perf` can be found in Appendix.

Improved Descriptions with Pragmatic Inference. In the case when $\texttt{perf}\,(\hat{e}, e_n) < \tau$, instead of greedily generating descriptions, we apply approximate pragmatic inference (Andreas & Klein, 2016; Fried et al., 2018) to improve the quality of descriptions. Intuitively, this approach emulates the teacher's ability to mentally simulate task execution before uttering descriptions. Assume that the teacher also implements its own execution policy, denoted by π_T, and has access to a simulator of the environment. A *pragmatic* execution-describing teacher is defined as $\mathbb{P}_T^{\text{prag}}(d \mid e) \propto \mathbb{P}_{\pi_T}(e \mid d)$. In words, this means that the more likely a request d invokes the teacher to generate an execution e, the more likely that the teacher describes e as d.

In practice, however, constructing the pragmatic teacher as above is often intractable. We follow Andreas et al. (2018), using a proposal distribution to generate a set of candidate descriptions, and then use $\mathbb{P}_{\pi_T}(e \mid d)$ to re-rank those candidates. Concretely, for execution $\hat{e}$ where $\texttt{perf}\,(\hat{e}, e_n) < \tau$, we use the language model $\tilde{\mathbb{P}}_T$ to generate a set of candidate descriptions $\mathcal{D}_{\text{cand}} = \left\{\hat{d}_{\text{greedy}}\right\} \cup \left\{\hat{d}_{\text{sample}}^{(k)}\right\}_{k=1}^{K}$. This set consists of the greedily decoded description $\hat{d}_{\text{greedy}} = \texttt{greedy}\left(\tilde{\mathbb{P}}_T(\cdot \mid \hat{e})\right)$ and K sample descriptions $\hat{d}_{\text{sample}}^{(k)} \sim \tilde{\mathbb{P}}_T(\cdot \mid \hat{e})$. We use the pairs $\left(e_n, d_n^{(j)}\right)$ of $\mathcal{B}_{\text{sim}}$ to train the teacher execution policy $\pi_T(a \mid s, d)$ using supervised learning. We select descriptions in $\mathcal{D}_{\text{cand}}$ from which π_T generates executions that incur high `perf` metrics:

$$\mathcal{D}_{\text{prag}} = \left\{d \mid d \in \mathcal{D}_{\text{cand}} \wedge \texttt{perf}\left(e^d, \hat{e}\right) \geq \tau\right\} \tag{5}$$

where $e^d \sim \mathbb{P}_{\pi_T}(\cdot \mid s_1, d)$, and s_1 is the start state of $\hat{e}$. The description is then chosen as follows:

$$\hat{d} \sim \begin{cases} \texttt{unif}\,(\mathcal{D}_{\text{prag}} \cup \{\emptyset\}) & \text{if } \texttt{perf}\,(\hat{e}, e_n) < \tau, \\ \texttt{unif}\,(\mathcal{D}_n) & \text{otherwise} \end{cases} \tag{6}$$

where $\emptyset$ is the empty string.

5.2. Vision-Language Navigation (NAV)

Problem and Environment. An agent executes natural language requests by navigating to locations in environments that photo-realistically emulate residential buildings (Anderson et al., 2018). Navigation in an environment is framed as traversing in a graph where each node represents a location and each edge connects two nearby unobstructed locations. A state s of an agent represents its location and the direction it is facing. In the beginning, the agent starts in state s_1 and receives a navigation request $d^\star$. At every time step, the agent is not given the true state s but only receives an observation o, which is a real-world RGB image capturing the panoramic view at its current location.

Agent Policy. The agent maintains a policy $\pi_\theta\,(a \mid o, d)$ that takes in a current observation o and a request d, and outputs an action $a \in V_{\text{adj}}$, where V_{adj} is set of locations that are adjacent to the agent's current location according to the environment graph. A special `<stop>` action is taken when the agent wants to terminate an episode or when it has taken H actions. The agent successfully fulfills a request if its final location is within three meters of the goal location.

Simulated Teacher. We simulate a teacher that does not know how the agent operates and thus cannot provide demonstrations. However, the teacher can verbally describe navigation executions. We follow §5.1, constructing a teacher $\mathbb{P}_T(d \mid e)$ that outputs language descriptions given executions $e = (o_0, a_1, \cdots, o_H)$.

5.3. Word Modification via Regular Expressions (REGEX)

Problem. A human gives an agent an input word w^{inp} and gives a natural language request $d^\star$ that asks it to modify the characters of the word. The agent must follow the request and output the modified word w^{out}. For example, given a word *embolden* and a request "*replace all n with c*", the correct output is *emboldec*. We train an agent that solves this problem via a semantic parsing approach. Given w^{inp} and $d^\star$, the agent generates a regular expression $\hat{a}_{1:H} = (\hat{a}_1, \cdots, \hat{a}_H)$, which is a sequence of characters. It then uses a regular expression compiler to apply the regular expression on the input word to produce an output word $\hat{w}^{\text{out}} = \texttt{compile}\left(w^{\text{inp}}, \hat{a}_{1:H}\right)$. Detail about the regular expression syntax is available in the Appendix.

Agent Policy and Environment. The agent maintains a

[3]Greedily decoding an RNN-based model refers to stepwise choosing the highest-probability class of the output softmax. In this case, the classes are words in the description vocabulary.

[4]The choice of `perf` only matters for experimentation and is not necessarily the same as the reward function R.

policy $\pi_\theta(a \mid s, d)$ that takes in a state s and a request d, and outputs distribution over characters $a \in V_{\text{regex}}$, where V_{regex} is the regular expression (character) vocabulary. A special `<stop>` action is taken when the agent wants to stop and apply the predicted regular expression onto the input word or when the length of the regular expression exceeds H. We set the initial state $s_1 = (w^{\text{inp}}, \emptyset)$, where $\emptyset$ is the empty string. A next state is determined as follows

$$s_{t+1} = \begin{cases} (\hat{w}^{\text{out}}, \hat{a}_{1:t}) & \text{if } \hat{a}_t = \texttt{<stop>}, \\ (w^{\text{inp}}, \hat{a}_{1:t}) & \text{otherwise} \end{cases} \tag{7}$$

where $\hat{w}^{\text{out}} = \texttt{compile}(w^{\text{inp}}, \hat{a}_{1:t})$. The agent successfully fulfills a request if the predicted output $\hat{w}^{\text{out}}$ exactly matches the ground-truth w^{out}.

Simulated Teacher. We demonstrate that a teacher without knowledge about regular expressions can teach the agent via language descriptions to solve this problem. Concretely, we model a teacher $P_T\left(d \mid \{w_j^{\text{inp}}, \hat{w}_j^{\text{out}}\}_{j=1}^{K}\right)$ that looks at K pairs of input and (predicted) output words and generates a description $\hat{d}$ that describes the transformation applied to the input words. The teacher requires multiple word pairs as input to reduce ambiguity. For example, *embolden* $\rightarrow$ *emboldec* can be described as "*replace all n with c*" or "*replace the last letter with c*". If the agent generates the output word by a regular expression that corresponds to the first description, observing more word pairs generated by the same regular expression (e.g. *now* $\rightarrow$ *cow*) increases the chance that the teacher outputs the first description. To generate K word pairs, in addition to the given input w^{inp}, the agent samples $K-1$ more input words from the dictionary and executes $d^\star$ on all K words using its policy.

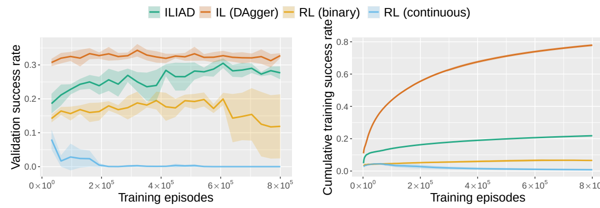
Importantly, we do *not* use any regular expression data in constructing the simulated teacher. The simulation data are tuples $\left(\mathcal{D}_n, \left(w_n^{\text{inp}}, w_n^{\text{out}}\right)\right)$ which are not annotated with ground-truth regular expressions. The teacher policy π_T is trained on these tuples to directly generate an output word instead of predicting a regular expression like the agent policy π_θ. Our setup illustrates a scenario where training with ILIAD is more scalable than imitation learning, which in this case requires experts that have knowledge about regular expressions.

5.4. Baselines and Evaluation Metrics

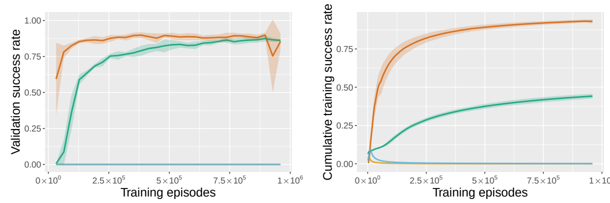
We compare interactive learning settings that employ different teaching media. Given an agent execution $\hat{e}$ of a request $d^\star$, we consider:

- *Learning from activity description* (ILIAD): the teacher returns a language description $\hat{d}$ of $\hat{e}$.
- *Imitation learning* (IL): the teacher demonstrates correct actions in the states that the agent visited, returning $e^\star = (s_1, a_1^\star, \cdots, a_H^\star, s_H)$, where s_i are the states in $\hat{e}$ and $a_i^\star$ is the optimal actions to take in those states.
- *Reinforcement learning* (RL): the teacher provides a scalar reward that evaluates $\hat{e}$. We consider a special case when rewards are provided only at the end of an episode. Because such feedback is cheap to collect (e.g., star ratings) (Nguyen et al., 2017b; Kreutzer et al., 2018; 2020), this setting is suitable for real-world applications. We experiment with both binary reward that indicates task success, and continuous reward that measures normalized distance to the goal (see Appendix).

We use ADEL in the ILIAD setting, DAgger (Ross et al., 2011) in IL, and REINFORCE[5] (Williams, 1992) in RL. We report the *success rates* of these algorithms, which are the fractions of examples on a held-data data split (validation or test) on which the agent successfully fulfills its requests.



(a) NAV



(b) REGEX

Figure 2. Success rate on validation set (average held-out return) and cumulative training success rate (average training return) over the course of training. For each algorithm, we report means and standard deviations over five runs with different random seeds.

6. Results

We compare learning algorithms not only on success rate, but also on the effort expended by the teacher. While task success rate is straightforward to compute, teacher effort is hard to quantify because it depends on many factors: the type of knowledge required to teach, the cognitive and physical abilities of the teacher, etc. For example, in REGEX, providing demonstration in forms of regular expressions may be easy for a computer science student, but could be

[5]We use a moving-average baseline to reduce variance. We also experimented with A2C (Mnih et al., 2016) but it was less stable in this sparse-reward setting. We are not aware of any work that successfully trains agents using RL without supervised-learning bootstrapping in the two problems we experiment on.

Table 2. Main results. We report means and standard deviations of success rates (%) over five runs with different random seeds. RL-Binary and RL-Cont refer to the RL settings with binary and continuous rewards, respectively. Sample complexity is the number of training episodes (or number of teacher responses) required to reach a validation success rate of at least c.

				Sample complexity ↓		
Learning setting	Algorithm	Val success rate (%) ↑	Test success rate (%) ↑	# Demonstrations	# Rewards	# Descriptions
Vision-language navigation					(c = 30.0%)	
IL	DAgger	35.6 ± 1.35	32.0 ± 1.63	45K ± 26K	-	-
RL-Binary	REINFORCE	22.4 ± 1.15	20.5 ± 0.58	-	+∞	-
RL-Cont	REINFORCE	11.1 ± 2.19	11.3 ± 1.25	-	+∞	-
ILIAD	ADEL	32.2 ± 0.97	31.9 ± 0.76	-	-	406K ± 31K
Word modification					(c = 85.0%)	
IL	DAgger	92.5 ± 0.53	93.0 ± 0.37	118K ± 16K	-	-
RL-Binary	REINFORCE	0.0 ± 0.00	0.0 ± 0.00	-	+∞	-
RL-Cont	REINFORCE	0.0 ± 0.00	0.0 ± 0.00	-	+∞	-
ILIAD	ADEL	88.1 ± 1.60	89.0 ± 1.30	-	-	573K ± 116K

Table 3. Effects of mixing with the approximate marginal. Sample complexity is measured with the same c values as in Table 2.

Training algorithm	Success rate (val)	Sample complexity
Vision-language navigation		
ADEL ($\lambda = 0.5$)	32.0	384K
ADEL, only approximate marginal ($\lambda = 1$)	29.4	+∞
ADEL, no approximate marginal ($\lambda = 0$)	0.0	+∞
Word modification		
ADEL ($\lambda = 0.5$)	88.0	608K
ADEL, only approximate marginal ($\lambda = 1$)	55.7	+∞
ADEL, no approximate marginal ($\lambda = 0$)	0.2	+∞

a challenge for someone who is unfamiliar with programming. In NAV, controlling a robot may not be viable for an individual with motor impairment, whereas generating language descriptions may infeasible for someone with a verbal-communication disorder. Because it is not possible cover all teacher demographics, our goal is to *quantitatively* compare the learning algorithms on learning effectiveness and efficiency, and *qualitatively* compare them on teacher effort to encode feedback intent.[6] Our overall findings (Table 1) highlight the strengths and weaknesses of each learning algorithm and can potentially aid practitioners in selecting algorithms that best suit their applications.

Main results. Our main results are given in Table 2. Overall, results in both problems match our expectations. The IL baseline achieves the highest success rates (on average, 35.6% on NAV and 92.5% on REGEX). This framework is the most effective way to teach the agents because it directly specifies the ground-truth actions while also teaching agents to recover from mistakes. The RL baseline is unable to reach competitive success rates; especially, in REGEX, the RL agent cannot learn the syntax of the regular expressions and completely fails at test time. ADEL's success rates are slightly lower than those of IL (3-4% lower than) but are substantially higher than those of RL (+9.8% on NAV and +88.1% on REGEX compared to the best RL results).

To measure learning efficiency, we report the number of teacher responses required to reach a reasonable success rate (30% for NAV and 85% for REGEX). We observe that all algorithms require hundreds of thousands of responses from the teacher to attain those success rates. The RL agents cannot learn effectively even after collecting more than 1M responses from the teachers. ADEL attains reasonable success rates using 5-8 times more responses than IL. This is a decent efficiency considering that ADEL needs to find the ground-truth executions in exponentially large search spaces, while IL directly communicates these executions to the agent. As ADEL lacks access to ground-truth executions, its average training returns are 2-4 times lower than those of IL (Figure 2).

Ablation. We conduct experiments to study the effectiveness of mixing with the approximate marginal in the ADEL algorithm (Table 3). First of all, we observe that learning cannot take off without the approximate marginal (λ = 0). On the other hand, using only the approximate marginal to generate executions degrades performance, in terms of both success rate and sample efficiency. This effect is more visible on REGEX where the success rate drops by 33% (compared to a 3% drop in NAV), indicating that the gap between the approximate marginal and the true marginal is larger in REGEX than in NAV. This matches our expectations because the space over valid executions in REGEX (i.e., the set of valid regular expressions generated from our template) has much more elements than the space over valid executions in NAV (i.e., the set of shortest-paths of lengths 2-6 in 54 graphs).

[6]The teacher's effort can be decomposed into (a) the effort to derive feedback intent (i.e. what it wants to convey to the agent) and (b) the effort to encode the intent (i.e. expressing it using the communication medium chosen by the learning protocol).

7. Conclusion

The communication protocol of a learning framework places natural boundaries on the learning efficiency of any algorithm that instantiates the framework. In this work, we illustrate the benefits of designing learning algorithms based on a natural, descriptive communication medium like human language. Employing such expressive protocols leads to ample room for improving learning algorithms. For future work, we will exploit the systematicity of language and the decomposability nature of tasks to reduce effort from feedback providers in these algorithms.

References

Agarwal, A., Kakade, S., Krishnamurthy, A., and Sun, W. Flambe: Structural complexity and representation learning of low rank mdps. In *Proceedings of Advances in Neural Information Processing Systems*, 2020.

Anderson, P., Wu, Q., Teney, D., Bruce, J., Johnson, M., Sünderhauf, N., Reid, I., Gould, S., and van den Hengel, A. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

Andreas, J. and Klein, D. Reasoning about pragmatics with neural listeners and speakers. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1173–1182, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1125. URL https://www.aclweb.org/anthology/D16-1125.

Andreas, J., Klein, D., and Levine, S. Learning with latent language. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 2166–2179, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1197. URL https://www.aclweb.org/anthology/N18-1197.

Artzi, Y., FitzGerald, N., and Zettlemoyer, L. Semantic parsing with combinatory categorial grammars. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Tutorials)*, pp. 2, Sofia, Bulgaria, August 2013. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/P13-5002.

Bisk, Y., Yuret, D., and Marcu, D. Natural language communication with robots. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 751–761, San Diego, California, June 2016. Association for Computational Linguistics. doi: 10.18653/v1/N16-1089. URL https://www.aclweb.org/anthology/N16-1089.

Chaplot, D. S., Sathyendra, K. M., Pasumarthi, R. K., Rajagopal, D., and Salakhutdinov, R. Gated-attention architectures for task-oriented language grounding. In *Association for the Advancement of Artificial Intelligence*, 2018.

Chen, D. L. and Mooney, R. J. Learning to interpret natural language navigation instructions from observations. In *Association for the Advancement of Artificial Intelligence*, 2011.

Chen, H., Suhr, A., Misra, D., and Artzi, Y. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019.

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

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

Das, A., Kottur, S., Moura, J. M., Lee, S., and Batra, D. Learning cooperative visual dialog agents with deep reinforcement learning. In *International Conference on Computer Vision*, 2017.

Evtimova, K., Drozdov, A., Kiela, D., and Cho, K. Emergent communication in a multi-modal, multi-step referential game. In *Proceedings of the International Conference on Learning Representations*, 2018.

Fried, D., Andreas, J., and Klein, D. Unified pragmatic models for generating and following instructions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pp. 1951–1963, New Orleans, Louisiana, June 2018. Association for Computational Linguistics. doi: 10.18653/v1/N18-1177. URL https://www.aclweb.org/anthology/N18-1177.

Fu, J., Korattikara, A., Levine, S., and Guadarrama, S. From language to goals: Inverse reinforcement learning for vision-based instruction following. In *Proceedings of the*

International Conference on Learning Representations, 2019.

Gaddy, D. and Klein, D. Pre-learning environment representations for data-efficient neural instruction following. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 1946–1956, Florence, Italy, July 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1188. URL https://www.aclweb.org/anthology/P19-1188.

Goldwasser, D. and Roth, D. Learning from natural instructions. *Machine learning*, 94(2):205–232, 2014.

Golland, D., Liang, P., and Klein, D. A game-theoretic approach to generating spatial descriptions. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pp. 410–419, Cambridge, MA, October 2010. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D10-1040.

Goodman, N. D. and Frank, M. C. Pragmatic language interpretation as probabilistic inference. *Trends in Cognitive Sciences*, 20(11):818–829, 2016.

Goyal, P., Niekum, S., and Mooney, R. J. Using natural language for reward shaping in reinforcement learning. In *International Joint Conference on Artificial Intelligence*, 2019.

Grice, H. P. Logic and conversation. In *Speech acts*, pp. 41–58. Brill, 1975.

Havrylov, S. and Titov, I. Emergence of language with multi-agent games: Learning to communicate with sequences of symbols. In *Proceedings of Advances in Neural Information Processing Systems*, 2017.

Hermann, K. M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., Szepesvari, D., Czarnecki, W., Jaderberg, M., Teplyashin, D., Wainwright, M., Apps, C., Hassabis, D., and Blunsom, P. Grounded language learning in a simulated 3D world. *CoRR*, abs/1706.06551, 2017.

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

Kottur, S., Moura, J., Lee, S., and Batra, D. Natural language does not emerge 'naturally' in multi-agent dialog. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 2962–2967, Copenhagen, Denmark, September 2017. Association for Computational Linguistics. doi: 10.18653/v1/D17-1321. URL https://www.aclweb.org/anthology/D17-1321.

Kreutzer, J., Sokolov, A., and Riezler, S. Bandit structured prediction for neural sequence-to-sequence learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1503–1513, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1138. URL https://www.aclweb.org/anthology/P17-1138.

Kreutzer, J., Uyheng, J., and Riezler, S. Reliability and learnability of human bandit feedback for sequence-to-sequence reinforcement learning. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1777–1788, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-1165. URL https://www.aclweb.org/anthology/P18-1165.

Kreutzer, J., Riezler, S., and Lawrence, C. Learning from human feedback: Challenges for real-world reinforcement learning in nlp. In *Proceedings of Advances in Neural Information Processing Systems*, 2020.

Labutov, I., Yang, B., and Mitchell, T. Learning to learn semantic parsers from natural language supervision. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 1676–1690, Brussels, Belgium, October-November 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1195. URL https://www.aclweb.org/anthology/D18-1195.

Langford, J. and Zhang, T. The epoch-greedy algorithm for multi-armed bandits with side information. In *Advances in neural information processing systems*, pp. 817–824, 2008a.

Langford, J. and Zhang, T. The epoch-greedy algorithm for multi-armed bandits with side information. In Platt, J., Koller, D., Singer, Y., and Roweis, S. (eds.), *Advances in Neural Information Processing Systems*, volume 20, pp. 817–824. Curran Associates, Inc., 2008b. URL https://proceedings.neurips.cc/paper/2007/file/4b04a686b0ad13dce35fa99fa4161c65-Paper.pdf.

Lazaridou, A., Peysakhovich, A., and Baroni, M. Multi-agent cooperation and the emergence of (natural) language. In *Proceedings of the International Conference on Learning Representations*, 2017.

Li, T. J.-J., Li, Y., Chen, F., and Myers, B. A. Programming iot devices by demonstration using mobile apps. In *International Symposium on End User Development*, pp. 3–17. Springer, 2017.

Li, T. J.-J., Chen, J., Mitchell, T. M., and Myers, B. A. Towards effective human-ai collaboration in gui-based interactive task learning agents. *Workshop on Artificial Intelligence for HCI: A Modern Approach (AI4HCI)*, 2020a.

Li, T. J.-J., Radensky, M., Jia, J., Singarajah, K., Mitchell, T. M., and Myers, B. A. Interactive task and concept learning from natural language instructions and gui demonstrations. In *The AAAI-20 Workshop on Intelligent Process Automation (IPA-20)*, 2020b.

Ling, H. and Fidler, S. Teaching machines to describe images via natural language feedback. In *Proceedings of Advances in Neural Information Processing Systems*, 2017.

Liu, C., Yang, S., Saba-Sadiya, S., Shukla, N., He, Y., Zhu, S.-C., and Chai, J. Jointly learning grounded task structures from language instruction and visual demonstration. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1482–1492, 2016.

MacGlashan, J., Babes-Vroman, M., desJardins, M., Littman, M. L., Muresan, S., Squire, S., Tellex, S., Arumugam, D., and Yang, L. Grounding english commands to reward functions. In *Robotics: Science and Systems*, 2015.

Magalhaes, G. I., Jain, V., Ku, A., Ie, E., and Baldridge, J. General evaluation for instruction conditioned navigation using dynamic time warping. In *Proceedings of Advances in Neural Information Processing Systems*, 2019.

Matuszek, C., FitzGerald, N., Zettlemoyer, L., Bo, L., and Fox, D. A joint model of language and perception for grounded attribute learning. In *Proceedings of the International Conference of Machine Learning*, 2012.

Mei, H., Bansal, M., and Walter, M. R. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2016.

Miryoosefi, S., Brantley, K., Daume III, H., Dudik, M., and Schapire, R. E. Reinforcement learning with convex constraints. In Wallach, H., Larochelle, H., Beygelzimer, A., dAlché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32, pp. 14093–14102. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper/2019/file/873be0705c80679f2c71fbf4d872df59-Paper.pdf.

Misra, D., Langford, J., and Artzi, Y. Mapping instructions and visual observations to actions with reinforcement learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2017.

Misra, D. K., Sung, J., Lee, K., and Saxena, A. Tell Me Dave: Context-sensitive grounding of natural language to mobile manipulation instructions. In *Robotics: Science and Systems (RSS)*, 2014.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In *Proceedings of the International Conference of Machine Learning*, 2016.

Monroe, W. and Potts, C. Learning in the Rational Speech Acts model. In *Proceedings of 20th Amsterdam Colloquium*, 2015.

Nguyen, K. and Daumé III, H. Help, anna! visual navigation with natural multimodal assistance via retrospective curiosity-encouraging imitation learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, November 2019. URL https://arxiv.org/abs/1909.01871.

Nguyen, K., Daumé III, H., and Boyd-Graber, J. Reinforcement learning for bandit neural machine translation with simulated human feedback. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1464–1474, Copenhagen, Denmark, September 2017a. Association for Computational Linguistics. doi: 10.18653/v1/D17-1153. URL https://www.aclweb.org/anthology/D17-1153.

Nguyen, K., Daumé III, H., and Boyd-Graber, J. Reinforcement learning for bandit neural machine translation with simulated human feedback. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pp. 1464–1474, Copenhagen, Denmark, September 2017b. Association for Computational Linguistics. doi: 10.18653/v1/D17-1153. URL https://www.aclweb.org/anthology/D17-1153.

Nguyen, K., Dey, D., Brockett, C., and Dolan, B. Vision-based navigation with language-based assistance via imitation learning with indirect intervention. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. URL https://arxiv.org/abs/1812.04155.

Pomerleau, D. A. Efficient training of artificial neural networks for autonomous navigation. *Neural Computation*, 3(1):88–97, 1991.

Ross, S., Gordon, G., and Bagnell, D. A reduction of imitation learning and structured prediction to no-regret online learning. In *Artificial Intelligence and Statistics (AISTATS)*, 2011.

Stadie, B. C., Abbeel, P., and Sutskever, I. Third-person imitation learning. In *Proceedings of the International Conference on Learning Representations*, 2017.

Sumers, T. R., Ho, M. K., Hawkins, R. D., Narasimhan, K., and Griffiths, T. L. Learning rewards from linguistic feedback. 2020.

Sun, W., Vemula, A., Boots, B., and Bagnell, J. A. Provably efficient imitation learning from observation alone. In *Proceedings of the International Conference of Machine Learning*, June 2019.

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

Tellex, S., Thaker, P., Joseph, J., and Roy, N. Toward learning perceptually grounded word meanings from unaligned parallel data. In *Proceedings of the Second Workshop on Semantic Interpretation in an Actionable Context*, pp. 7–14, Montréal, Canada, June 2012. Association for Computational Linguistics. URL https://www.aclweb.org/anthology/W12-2802.

Wang, S. I., Liang, P., and Manning, C. D. Learning language games through interaction. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2368–2378, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1224. URL https://www.aclweb.org/anthology/P16-1224.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8, 1992.

Winograd, T. Understanding natural language. *Cognitive Psychology*, 3(1):1–191, 1972.

Notation	Definition
$\Delta(\mathcal{U})$	Space of all distributions over a set $\mathcal{U}$
$\mathtt{unf}(\mathcal{U})$	Denotes the uniform distribution over a set $\mathcal{U}$
$\\|.\\|_p$	p-norm
$\mathrm{D}_{\mathrm{KL}}(Q_1(x \mid y) \mid\mid Q_2(x \mid y))$	KL-divergence between two distributions $Q_1(\cdot \mid y)$ and $Q_2(\cdot \mid y)$ over a countable set $\mathcal{X}$. Formally, $\mathrm{D}_{\mathrm{KL}}(Q_1(x \mid y) \mid\mid Q_2(x \mid y)) = \sum_{x \in \mathcal{X}} Q_1(x \mid y) \ln \frac{Q_1(x\mid y)}{Q_2(x\mid y)}$.
$\mathtt{supp}\ Q(x)$	Support of a distribution $Q \in \Delta(\mathcal{X})$. Formally, $\mathtt{supp}Q(x) = \{x \in \mathcal{X} \mid Q(x) > 0\}$.
$\mathbb{N}$	Set of natural numbers
$\mathcal{S}$	State space
s	A single state in $\mathcal{S}$
$\mathcal{A}$	Finite action space
a	a single action in $\mathcal{A}$
$\mathcal{D}$	Set of all possible descriptions and requests
d	A single description or request
$T : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$	Transition function with $T(s' \mid s, a)$ denoting the probability of transitioning to state s' given state s and action a.
$\mathcal{R}$	Family of reward functions
$R : \mathcal{S} \times \mathcal{A} \to [0, 1]$	Reward function with $R(s, a)$ denoting the reward for taking action a in state s
H	Horizon of the problem denoting the number of actions in a single episode.
e	An execution $e = (s_1, a_1, s_2, \cdots, s_H, a_H)$ describing states and actions in an episode.
$q = (R, d, s_1)$	A single task comprising of reward function R, request d and start state s_1
$\mathbb{P}^\star(q)$	Task distribution defined by the world
$\mathbb{P}^\star(e, R, s, d)$	Joint distribution over executions and task (see Equation 2).
$\mathbb{P}_T(d \mid e)$	Teacher model denoting distribution over descriptions d for a given execution e.
Θ	Set of all parameters of agent's policy.
θ	Parameters of agent's policy. Belongs to the set Θ.
$\pi_\theta(a \mid s, d)$	Agent's policy denoting the probability of action a given state s, description d, and parameters θ.

Table 4. List of common notations and their definitions.

Appendix: Interactive Learning from Activity Description

The appendix is organized as follows:

- Theoretical guarantees for ADEL are stated and proved in Appendix A;
- Detailed comparison with related works that use language feedback (Appendix B);
- Settings of the two problems we conduct experiments on (Appendix C);
- A practical implementation of the ADEL algorithm that we use for experimentation (Appendix D);
- Training details including model architecture and hyperparameters (Appendix E);
- Qualitative examples (Appendix F).

We provide a list of notations in Table 4 on page 13.

A. Theoretical Analysis of ADEL

In this section, we provide a theoretical justification for an epoch-version of ADEL for the case of $H = 1$. We prove consistency results showing ADEL learns a near-optimal policy, and we also derive the convergence rate under the assumption that we perform maximum likelihood estimation optimally and the teacher is consistent. We call a teacher model $\mathbb{P}_T(d \mid e)$ to be consistent if for every execution e and description d we have $\mathbb{P}_T(d \mid e) = \mathbb{P}^\star(d \mid e)$. Recall that the conditional distribution $\mathbb{P}^\star(d \mid e)$ is derived from the joint distribution defined in Equation 2. We will use superscript * to denote all probability distributions that are derived from this joint distribution.

We start by writing the epoch-version of ADEL in Algorithm 4 for an arbitrary value of H. The epoch version of ADEL runs an outer loop of epochs (line 3-10). The agent model is updated only at the end of an epoch. In the inner loop (line 5-9), the agent samples a batch using the teacher model and the agent model. This is used to update the model at the end of the epoch.

At the start of the n^{th} epoch, our sampling scheme in line 6-9 defines a procedure to sample $(\hat{e}, \hat{d})$ from a distribution D_n that remains fixed over this whole epoch. To define D_n, we first define $\mathbb{P}_n(e) = \mathbb{E}_{(R,d,s_1)\sim\mathbb{P}^\star(q)}\left[\mathbb{P}_n(e \mid s_1, d)\right]$ where we use the shorthand $\mathbb{P}_n(e \mid s_1, d)$ to refer to $\mathbb{P}_{\pi_{\theta_n}}(e \mid s_1, d)$. Note that $\hat{e} \sim \mathbb{P}_n(e)$ in line 7. As $\hat{d} \sim \mathbb{P}^\star(d \mid \hat{e})$, therefore, we arrive at the following form of D_n:

$$D_n(\hat{e}, \hat{d}) = \mathbb{P}^\star(\hat{d} \mid \hat{e})\mathbb{P}_n(\hat{e}). \tag{8}$$

We will derive our theoretical guarantees for $H = 1$. This setting is known as the contextual bandit setting (Langford & Zhang, 2008a), and while simpler than general reinforcement learning setting, it captures a large non-trivial class of problems. In this case, an execution $e = [s_1, a_1]$ can be described by the start state s_1 and a single action $a_1 \in \mathcal{A}$ taken by the agent. Since there is a single state and action in any execution, therefore, for cleaner notations we will drop the subscript and simply write s, a instead of s_1, a_1. For convenience, we also define a few extra notations. Firstly, we define the marginal distribution $D_n(s, \hat{d}) = \sum_{a' \in \mathcal{A}} D_n([s, a'], d)$. Secondly, let $\mathbb{P}^\star(s)$ be the marginal distribution over start state s given by $\mathbb{E}_{(R,d,s_1)\sim\mathbb{P}^\star(q)}[\mathbf{1}\{s_1 = s\}]$. We state some useful relations between these probability distributions in the next lemma.

Algorithm 4 EPOCHADEL: Epoch Version of ADEL. We assume the teacher is consistent, i.e., $\mathbb{P}_T(d \mid e) = \mathbb{P}^\star(d \mid e)$ for every (d, e).

1: **Input**: teacher model $\mathbb{P}^\star(d \mid e)$ and task distribution model $\mathbb{P}^\star(q)$.
2: Initialize agent policy $\pi_{\theta_1} : \mathcal{S} \times \mathcal{D} \to \mathtt{unf}(\mathcal{A})$
3: **for** $n = 1, 2, \cdots, N$ **do**
4: $\quad \mathcal{B} = \emptyset$
5: $\quad$ **for** $m = 1, 2, \cdots, M$ **do**
6: $\quad\quad$ World samples $q = (R, d^\star, s_1) \sim \mathbb{P}^\star(\cdot)$
7: $\quad\quad$ Agent generates $\hat{e} \sim \mathbb{P}_{\pi_{\theta_n}}(\cdot \mid s_1, d^\star)$
8: $\quad\quad$ Teacher generates description $\hat{d} \sim \mathbb{P}^\star(\cdot \mid \hat{e})$
9: $\quad\quad$ $\mathcal{B} \leftarrow \mathcal{B} \cup \left\{\left(\hat{e}, \hat{d}\right)\right\}$
10: $\quad$ Update agent policy using batch updates:

$$\theta_{n+1} \leftarrow \arg\max_{\theta' \in \Theta} \sum_{(\hat{e},\hat{d})\in\mathcal{B}} \sum_{(s,a_s)\in\hat{e}} \log \pi_{\theta'}(a_s \mid s, \hat{d})$$

where a_s is the action taken by the agent in state s in execution $\hat{e}$.
return π_θ

Lemma 2. *For any $n \in \mathbb{N}$, we have:*

$$\mathbb{P}_n(e := [s, a]) = \mathbb{P}^\star(s)\mathbb{P}_n(a \mid s), \text{ where } \mathbb{P}_n(a \mid s) := \sum_d \mathbb{P}^\star(d \mid s)\mathbb{P}_n(a \mid s, d). \tag{9}$$

Proof. We first compute the marginal distribution $\sum_{a' \in \mathcal{A}} \mathbb{P}_n(e' := [s, a'])$ over s:

$$\sum_{a' \in \mathcal{A}} \mathbb{P}_n(e' := [s, a']) = \sum_{a' \in \mathcal{A}} \sum_{R,d} \mathbb{P}^\star(R, d, s)\mathbb{P}_n(a' \mid s, d) = \sum_{R,d} \mathbb{P}^\star(R, d, s) = \mathbb{P}^\star(s).$$

Next we compute the conditional distribution $\mathbb{P}_n(a \mid s)$ as shown:

$$\mathbb{P}_n(a \mid s) = \frac{\mathbb{P}_n([s, a])}{\sum_{a' \in \mathcal{A}} \mathbb{P}_n([s, a'])} = \sum_{R,d} \frac{\mathbb{P}^\star(R, d, s)\mathbb{P}_n(a \mid s, d)}{\mathbb{P}^\star(s)} = \sum_d \frac{\mathbb{P}^\star(s, d)\mathbb{P}_n(a \mid s, d)}{\mathbb{P}^\star(s)} = \sum_d \mathbb{P}^\star(d \mid s)\mathbb{P}_n(a \mid s, d).$$

This also proves $\mathbb{P}_n([s,a]) = \mathbb{P}^\star(s)\mathbb{P}_n(a \mid s)$. □

For $H = 1$, the update equation in line 10 solves the following optimization equation:

$$\max_{\theta' \in \Theta} J_n(\theta) \qquad \text{where} \quad J_n(\theta) := \sum_{(\hat{e}:=[s,a],\hat{d}) \in \mathcal{B}} \ln \pi_{\theta'}(a \mid s, \hat{d}). \tag{10}$$

Here $J_n(\theta)$ is the empirical objective whose expectation over draws of batches is given by:

$$\mathbb{E}[J_n(\theta)] = \mathbb{E}_{(\hat{e}=[s,a],d)\sim D_n}\left[\ln \pi_\theta(a \mid s, d)\right].$$

As this is negative of the cross entropy loss, the Bayes optimal value would be achieved for $\pi_\theta(a \mid s, d) = D_n(a \mid s, d)$ for all $a \in \mathcal{A}$ and every $(s,d) \in \mathtt{supp} D_n(s,d)$. We next state the form of this Bayes optimal model and then state our key realizability assumption.

Lemma 3. *Fix $n \in \mathbb{N}$. For every $(s,d) \in \mathtt{supp} D_n(s,d)$ the value of the Bayes optimal model $D_n(a \mid s,d)$ at the end of the n^{th} epoch is given by:*

$$D_n(a \mid s, d) = \frac{\mathbb{P}^\star(d \mid [s,a])\mathbb{P}_n(a \mid s)}{\sum_{a' \in \mathcal{A}} \mathbb{P}^\star(d \mid [s,a'])\mathbb{P}_n(a' \mid s)}.$$

Proof. The Bayes optimal model is given by $D_n(a \mid s,d)$ for every $(s,d) \in \mathtt{supp} D_n(s,d)$. We compute this using Bayes' theorem.

$$D_n(a \mid s,d) = \frac{D_n([s,a],d)}{\sum_{a' \in \mathcal{A}} D_n([s,a'],d)} = \frac{\mathbb{P}^\star(d \mid [s,a])\mathbb{P}_n([s,a])}{\sum_{a' \in \mathcal{A}} \mathbb{P}^\star(d \mid [s,a'])\mathbb{P}_n([s,a'])} = \frac{\mathbb{P}^\star(d \mid [s,a])\mathbb{P}_n(a \mid s)}{\sum_{a' \in \mathcal{A}} \mathbb{P}^\star(d \mid [s,a'])\mathbb{P}_n(a' \mid s)}.$$

The last equality above uses Lemma 2. □

In order to learn the Bayes optimal model, we need our policy class to be expressive enough to contain this model. We formally state this *realizability* assumption below.

Assumption 1 (Realizability)**.** *For every $\theta \in \Theta$, there exists $\theta' \in \Theta$ such that for every start state s, description d we have:*

$$\forall a \in \mathcal{A}, \quad \pi_{\theta'}(a \mid s, d) = \frac{\mathbb{P}^\star(d \mid [s,a])Q_\theta(a \mid s)}{\sum_{a' \in \mathcal{A}} \mathbb{P}^\star(d \mid [s,a'])Q_\theta(a' \mid s)}, \quad \textit{where} \quad Q_\theta(a \mid s) = \sum_{d'} \mathbb{P}^\star(d' \mid s)\pi_\theta(a \mid s, d').$$

We can use the realizability assumption along with convergence guarantees for log-loss to state the following result:

Theorem 4 (Theorem 21 of (Agarwal et al., 2020))**.** *Fix $m \in \mathbb{N}$ and $\delta \in (0,1)$. Let $\{(d^{(i)}, e^{(i)} = [s^{(i)}, a^{(i)}]\}_{i=1}^m$ be i.i.d draws from $D_n(e,d)$ and let θ_{n+1} be the solution to the optimization problem in line 10 of the n^{th} epoch of* EPOCHADEL*. Then with probability at least $1-\delta$ we have:*

$$\mathbb{E}_{s,d\sim D_n}\left[\|D_n(a \mid s,d) - \mathbb{P}_{\pi_{\theta_{n+1}}}(a \mid s,d)\|_1\right] \le C\sqrt{\frac{1}{m}\ln |\Theta|/\delta}, \tag{11}$$

where $C > 0$ is a universal constant.

Please see Agarwal et al. (2020) for a proof. Lemma 4 implies that assuming realizability, as $M \to \infty$, our learned solution converges to the Bayes optimal model pointwise on the support over $D_n(s,d)$. Since we are only interested in consistency, we will assume $M \to \infty$ and assume $\mathbb{P}_{n+1}(a \mid s,d) = D_n(a \mid s,d)$ for every $(s,d) \in \mathtt{supp} D_n(s,d)$. We will refer to this as optimally performing the maximum likelihood estimation at n^{th} epoch. If the learned policy is given by $\mathbb{P}_{n+1}(a \mid s,d) = D_n(a \mid s,d)$, then the next Lemma states the relationship between the marginal distribution $\mathbb{P}_{n+1}(a \mid s)$ for the next time epoch and marginal $\mathbb{P}_n(a \mid s)$ for this epoch.

Lemma 5 (Inductive Relation Between Marginals)**.** *For any $n \in \mathbb{N}$, if we optimally perform the maximum likelihood estimation at the n^{th} epoch of* EPOCHADEL*, then for all start states s, the marginal distribution $\mathbb{P}_{n+1}(a \mid s)$ for the $(n+1)^{th}$ epoch is given by:*

$$\mathbb{P}_{n+1}(a \mid s) = \sum_d \frac{\mathbb{P}^\star(d \mid [s,a])\mathbb{P}_n(a \mid s)\mathbb{P}^\star(d \mid s)}{\sum_{a' \in \mathcal{A}} \mathbb{P}^\star(d \mid [s,a'])\mathbb{P}_n(a' \mid s)}.$$

Proof. The proof is completed as follows:

$$\mathbb{P}_{n+1}(a \mid s) = \sum_d \mathbb{P}^\star(d \mid s)\mathbb{P}_{n+1}(a \mid s, d) = \sum_d \frac{\mathbb{P}^\star(d \mid [s,a])\mathbb{P}_n(a \mid s)\mathbb{P}^\star(d \mid s)}{\sum_{a' \in \mathcal{A}} \mathbb{P}^\star(d \mid [s,a'])\mathbb{P}_n(a' \mid s)},$$

where the first step uses Lemma 2 and the second step uses $\mathbb{P}_{n+1}(a \mid s, d) = D_n(a \mid s, d)$ (optimally solving maximum likelihood) and the form of D_n from Lemma 3. □

A.1. Proof of Convergence for Marginal Distribution

Our previous analysis associates a probability distribution $\mathbb{P}_n(a \mid s, d)$ and $\mathbb{P}_n(a \mid s)$ with the n^{th} epoch of EPOCHADEL. For any $n \in \mathbb{N}$, the n^{th} epoch of EPOCHADEL can be viewed as a transformation of $\mathbb{P}_n(a \mid s, d) \mapsto \mathbb{P}_{n+1}(a \mid s, d)$ and $\mathbb{P}_n(a \mid s) \mapsto \mathbb{P}_{n+1}(a \mid s)$. In this section, we show that under certain conditions, the running average of the marginal distributions $\mathbb{P}_n(a \mid d)$ converges to the optimal marginal distribution $\mathbb{P}^\star(a \mid d)$. We then discuss how this can be used to learn the optimal policy $\mathbb{P}^\star(a \mid s, d)$.

We use a potential function approach to measure the progress of each epoch. Specifically, we will use KL-divergence as our choice of potential function. The next lemma bounds the change in potential after a single iteration.

Lemma 6. *[Potential Difference Lemma] For any $n \in \mathbb{N}$ and start state s, we define the following distribution over descriptions $\mathbb{P}_n(d \mid s) := \sum_{a' \in \mathcal{A}} \mathbb{P}^\star(d \mid [s,a])\mathbb{P}_n(a \mid s)$. Then for every start state s we have:*

$$\mathrm{D}_{\mathrm{KL}}(\mathbb{P}^\star(a \mid s) \mid\mid \mathbb{P}_{n+1}(a \mid s)) - \mathrm{D}_{\mathrm{KL}}(\mathbb{P}^\star(a \mid s) \mid\mid \mathbb{P}_n(a \mid s)) \leq -\mathrm{D}_{\mathrm{KL}}(\mathbb{P}^\star(d \mid s) \mid\mid \mathbb{P}_n(d \mid s)).$$

Proof. The change in potential from the start of n^{th} epoch to its end is given by:

$$\mathrm{D}_{\mathrm{KL}}(\mathbb{P}^\star(a \mid s) \mid\mid \mathbb{P}_{n+1}(a \mid s)) - \mathrm{D}_{\mathrm{KL}}(\mathbb{P}^\star(a \mid s) \mid\mid \mathbb{P}_n(a \mid s)) = -\sum_{a \in \mathcal{A}} \mathbb{P}^\star(a \mid s) \ln\left(\frac{\mathbb{P}_{n+1}(a \mid s)}{\mathbb{P}_n(a \mid s)}\right) \tag{12}$$

Using Lemma 5 and the definition of $\mathbb{P}_n(d \mid s)$ we get:

$$\frac{\mathbb{P}_{n+1}(a \mid s)}{\mathbb{P}_n(a \mid s)} = \sum_d \frac{\mathbb{P}^\star(d \mid [s,a])\mathbb{P}^\star(d \mid s)}{\sum_{a' \in \mathcal{A}} \mathbb{P}^\star(d \mid [s,a'])\mathbb{P}_n(a' \mid s)} = \sum_d \mathbb{P}^\star(d \mid [s,a])\frac{\mathbb{P}^\star(d \mid s)}{\mathbb{P}_n(d \mid s)}.$$

Taking logarithms and applying Jensen's inequality gives:

$$\ln\left(\frac{\mathbb{P}_{n+1}(a \mid s)}{\mathbb{P}_n(a \mid s)}\right) = \ln\left(\sum_d \mathbb{P}^\star(d \mid [s,a])\frac{\mathbb{P}^\star(d \mid s)}{\mathbb{P}_n(d \mid s)}\right) \geq \sum_d \mathbb{P}^\star(d \mid [s,a]) \ln\left(\frac{\mathbb{P}^\star(d \mid s)}{\mathbb{P}_n(d \mid s)}\right). \tag{13}$$

Taking expectations of both sides with respect to $\mathbb{P}^\star(a \mid s)$ gives us:

$$\begin{aligned}\sum_a \mathbb{P}^\star(a \mid s) \ln\left(\frac{\mathbb{P}_{n+1}(a \mid s)}{\mathbb{P}_n(a \mid s)}\right) &\geq \sum_a \sum_d \mathbb{P}^\star(a \mid s)\mathbb{P}^\star(d \mid [s,a]) \ln\left(\frac{\mathbb{P}^\star(d \mid s)}{\mathbb{P}_n(d \mid s)}\right)\\ &= \sum_d \mathbb{P}^\star(d \mid s) \ln\left(\frac{\mathbb{P}^\star(d \mid s)}{\mathbb{P}_n(d \mid s)}\right)\\ &= \mathrm{D}_{\mathrm{KL}}(\mathbb{P}^\star(d \mid s) \mid\mid \mathbb{P}_n(d \mid s))\end{aligned}$$

where the last step uses the definition of $\mathbb{P}_n(d \mid s)$. The proof is completed by combining the above result with Equation 12. □

The $\mathbb{P}_s$ matrix. For a fixed start state s, we define $\mathbb{P}_s$ as the matrix whose entries are $\mathbb{P}^\star(d \mid [s,a])$. The columns of this matrix range over actions, and the rows range over descriptions. We denote the minimum singular value of the description matrix $\mathbb{P}_s$ by $\sigma_{min}(s)$.

We state our next assumption that the minimum singular value of $\mathbb{P}_s$ matrix is non-zero.

Assumption 2 (Minimum Singular Value is Non-Zero)**.** *For every start state s, we assume $\sigma_{min}(s) > 0$.*

Intuitively, this assumption states that there is enough information in the descriptions for the agent to decipher probabilities over actions from learning probabilities over descriptions. More formally, we are trying to decipher $\mathbb{P}^\star(a \mid s)$ using access to two distributions: $\mathbb{P}^\star(d \mid s)$ which generates the initial requests, and the teacher model $\mathbb{P}^\star(d \mid [s, a])$ which is used to describe an execution $e = [s, a]$. This can result in an underspecified problem. The only constraints these two distributions place on $\mathbb{P}^\star(a \mid s)$ is that $\sum_{a \in \mathcal{A}} \mathbb{P}^\star(d \mid [s, a])\mathbb{P}^\star(a \mid s) = \mathbb{P}^\star(d \mid s)$. This means all we know is that $\mathbb{P}^\star(a \mid s)$ belongs to the following set of solutions of the previous linear systems of equation:

$$\left\{ Q(a \mid s) \mid \sum_{a \in \mathcal{A}} \mathbb{P}^\star(d \mid [s, a]) Q(a \mid s) = \mathbb{P}^\star(d \mid s) \; \forall d, \qquad Q(a \mid s) \text{ is a distribution} \right\}.$$

As $\mathbb{P}^\star(a \mid s)$ belongs to this set hence this set is nonempty. However, if we also assume that $\sigma_{min}(s) > 0$ then the above set has a unique solution. Recall that singular values are square root of eigenvalues of $\mathbb{P}_s^\top \mathbb{P}_s$, and so $\sigma_{min}(s) > 0$ implies that the matrix $\mathbb{P}_s^\top \mathbb{P}_s$ is invertible. [7] This means, we can find the unique solution of the linear systems of equation by multiplying both sides by $(\mathbb{P}_s^\top \mathbb{P}_s)^{-1} \mathbb{P}_s^\top$. Hence, Assumption 2 makes it possible for us to find $\mathbb{P}^\star(a \mid s)$ using just the information we have. Note that we cannot solve the linear system of equations directly since the description space and action space can be extremely large. Hence, we use an oracle based solution via reduction to supervised learning.

The next theorem shows that the running average of learned probabilities $\mathbb{P}_n(a \mid s)$ converges to the optimal marginal distribution $\mathbb{P}^\star(a \mid s)$ at a rate determined by the inverse square root of the number of epochs of ADEL, the minimum singular value of the matrix $\mathbb{P}_s$, and the KL-divergence between optimal marginal and initial value.

Theorem 7. *[Rate of Convergence for Marginal] For any $t \in \mathbb{N}$ we have:*

$$\|\mathbb{P}^\star(a \mid s) - \frac{1}{t} \sum_{n=1}^{t} \mathbb{P}_n(a \mid s)\|_2 \le \frac{1}{\sigma_{min}(s)} \sqrt{\frac{2}{t} \mathrm{D}_{\mathrm{KL}}(\mathbb{P}^\star(a \mid s) \mid\mid \mathbb{P}_1(a \mid s))},$$

and if $\mathbb{P}_1(a \mid s, d)$ is a uniform distribution for every s and d, then

$$\|\mathbb{P}^\star(a \mid s) - \frac{1}{t} \sum_{n=1}^{t} \mathbb{P}_n(a \mid s)\|_2 \le \frac{1}{\sigma_{min}(s)} \sqrt{\frac{2 \ln |\mathcal{A}|}{t}}.$$

Proof. We start with Lemma 6 and bound the right hand side as shown:

$$\begin{aligned}
\mathrm{D}_{\mathrm{KL}}(\mathbb{P}^\star(a \mid s) \mid\mid \mathbb{P}_{n+1}(a \mid s)) - \mathrm{D}_{\mathrm{KL}}(\mathbb{P}^\star(a \mid s) \mid\mid \mathbb{P}_n(a \mid s)) &\le -\mathrm{D}_{\mathrm{KL}}(\mathbb{P}^\star(d \mid s) \mid\mid \mathbb{P}_n(d \mid s)) \\
&\le -\frac{1}{2} \|\mathbb{P}^\star(d \mid s) - \mathbb{P}_n(d \mid s)\|_1^2, \\
&\le -\frac{1}{2} \|\mathbb{P}^\star(d \mid s) - \mathbb{P}_n(d \mid s)^2\|_2^2 \\
&= -\frac{1}{2} \|\mathbb{P}_s \left\{ \mathbb{P}^\star(a \mid s) - \mathbb{P}_n(a \mid s) \right\} \|_2^2, \\
&\le -\frac{1}{2} \sigma_{min}(s)^2 \|\mathbb{P}^\star(a \mid s) - \mathbb{P}_n(a \mid s)\|_2^2,
\end{aligned}$$

where the second step uses Pinsker's inequality. The third step uses the property of p-norms, specifically, $\|\nu\|_2 \le \|\nu\|_1$ for all ν. The fourth step, uses the definition of $\mathbb{P}^\star(d \mid s) = \sum_{a' \in \mathcal{A}} \mathbb{P}(d \mid s, a') \mathbb{P}^\star(a' \mid s))$ and $\mathbb{P}_n(d \mid s) = \sum_{a' \in \mathcal{A}} \mathbb{P}(d \mid s, a') \mathbb{P}_n(a' \mid s)$. We interpret the notation $\mathbb{P}^\star(a \mid s)$ as a vector over actions whose value is the probability $\mathbb{P}^\star(a \mid s)$. Therefore, $\mathbb{P}_s \mathbb{P}^\star(a \mid s)$ represents a matrix-vector multiplication. Finally, the last step, uses $\|Ax\|_2 \ge \sigma_{min}(A) \|x\|_2$ for any vector x and matrix A of compatible shape such that Ax is defined, where $\sigma_{min}(A)$ is the smallest singular value of A.

Summing over n from $n = 1$ to t and rearranging the terms we get:

$$\mathrm{D}_{\mathrm{KL}}(\mathbb{P}^\star(a \mid s) \mid\mid \mathbb{P}_{t+1}(a \mid s)) \le \mathrm{D}_{\mathrm{KL}}(\mathbb{P}^\star(a \mid s) \mid\mid \mathbb{P}_1(a \mid s)) - \frac{1}{2} \sigma_{min}(s)^2 \sum_{n=1}^{t} \|\mathbb{P}^\star(a \mid s) - \mathbb{P}_n(a \mid s)\|_2^2.$$

[7]Recall that a matrix of the form $A^\top A$ always have non-negative eigenvalues.

As the left hand-side is positive we get:

$$\sum_{n=1}^{t} \|\mathbb{P}^\star(a \mid s) - \mathbb{P}_n(a \mid s)\|_2^2 \leq \frac{2}{\sigma_{min}(s)^2} \mathrm{D}_{\mathrm{KL}}(\mathbb{P}^\star(a \mid s) \mid\mid \mathbb{P}_1(a \mid s)).$$

Dividing by t and applying Jensen's inequality (specifically, $\mathbb{E}[X^2] \geq \mathbb{E}[|X|]^2$) we get:

$$\frac{1}{t} \sum_{n=1}^{t} \|\mathbb{P}^\star(e) - \mathbb{P}_n(e)\|_2 \leq \frac{1}{\sigma_{min}(s)} \sqrt{\frac{2}{t} \mathrm{D}_{\mathrm{KL}}(\mathbb{P}^\star(a \mid s) \mid\mid \mathbb{P}_1(a \mid s))} \tag{14}$$

Using the triangle inequality, the left hand side can be bounded as:

$$\frac{1}{t} \sum_{n=1}^{t} \|\mathbb{P}^\star(a \mid s) - \mathbb{P}_n(a \mid s)\|_2 \geq \|\mathbb{P}^\star(a \mid s) - \frac{1}{t} \sum_{n=1}^{t} \mathbb{P}_n(a \mid s)\|_2 \tag{15}$$

Combining the previous two equations proves the main result. Finally, note that if $\mathbb{P}_1(a \mid s, d) = {}^1/_{|\mathcal{A}|}$ for every value of s, d, and a, then $\mathbb{P}_1(a \mid s)$ is also a uniform distribution over actions. The initial KL-divergence is then bounded by $\ln |\mathcal{A}|$ as shown below:

$$\mathrm{D}_{\mathrm{KL}}(\mathbb{P}^\star(a \mid s) \mid\mid \mathbb{P}_1(a \mid s)) = -\sum_{a \in \mathcal{A}} \mathbb{P}^\star(a \mid s) \ln \frac{1}{|\mathcal{A}|} + \sum_{a \in \mathcal{A}} \mathbb{P}^\star(a \mid s) \ln \mathbb{P}^\star(a \mid s) \leq \ln |\mathcal{A}|,$$

where the second step uses the fact that entropy of a distribution is non-negative. This completes the proof. □

A.2. Proof of Convergence to Near-Optimal Policy

Finally, we discuss how to learn $\mathbb{P}^\star(a \mid s, d)$ once we learn $\mathbb{P}^\star(a \mid s)$. Since we only derive convergence of running average of $\mathbb{P}_n(a \mid s)$ to $\mathbb{P}^\star(a \mid s)$, therefore, we cannot expect $\mathbb{P}_n(a \mid s, d)$ to converge to $\mathbb{P}^\star(a \mid s, d)$. Instead, we will show that if we perform line 4-10 in Algorithm 4 using the running average of policies, then the learned Bayes optimal policy will converge to the near-optimal policy. The simplest way to accomplish this with Algorithm 4 is to perform the block of code in line 4-10 twice, once when taking actions according to $\mathbb{P}_n(a \mid s, d)$, and once when taking actions according to running average policy $\tilde{\mathbb{P}}_n(a \mid s, d) = \frac{1}{n} \sum_{t=1}^{n} \tilde{\mathbb{P}}_t(a \mid s, d)$. This will give us two Bayes optimal policy in 10 one each for the current policy $\mathbb{P}_n(a \mid s, d)$ and the running average policy $\tilde{\mathbb{P}_n}(a \mid s, d)$. We use the former for roll-in in the future and the latter for evaluation on held-out test set.

For convenience, we first define an operator that denotes mapping of one agent policy to another.

W **operator.** Let $\mathbb{P}(a \mid s, d)$ be an agent policy used to generate data in any epoch of EpochAdel (line 5-9). We define the W operator as the mapping to the Bayes optimal policy for the optimization problem solved by EpochAdel in line 10 which we denote by $(W\mathbb{P})$. Under the realizability assumption (Assumption 1), the agent learns the $W\mathbb{P}$ policy when $M \to \infty$. Using Lemma 2 and Lemma 3, we can verify that:

$$(W\mathbb{P})(a \mid s, d) = \frac{\mathbb{P}^\star(d \mid [s, a])\mathbb{P}(a \mid s)}{\sum_{a' \in \mathcal{A}} \mathbb{P}^\star(d \mid [s, a'])\mathbb{P}(a' \mid s)}, \quad \text{where} \quad \mathbb{P}(a \mid s) = \sum_{d} \mathbb{P}^\star(d \mid s)\mathbb{P}(a \mid s, d).$$

We first show that our operator is smooth around $\mathbb{P}^\star(a \mid s)$.

Lemma 8 (Smoothness of W)**.** *For any start state s and description $d \in \texttt{supp}\, \mathbb{P}^\star(d \mid s)$, there exists a finite constant K_s such that:*

$$\|W\mathbb{P}(a \mid s, d) - W\mathbb{P}^\star(a \mid s, d)\|_1 \leq K_s \|\mathbb{P}(a \mid s) - \mathbb{P}^\star(a \mid s)\|_1.$$

Proof. We define $\mathbb{P}(d \mid s) = \sum_{a' \in \mathcal{A}} \mathbb{P}^\star(d \mid s, a')\mathbb{P}(a' \mid s)$. Then from the definition of operator W we have:

$$
\begin{aligned}
&|W\mathbb{P}(a \mid s, d) - W\mathbb{P}^\star(a \mid s, d)|_1 \\
&= \sum_{a \in \mathcal{A}} \left| \frac{\mathbb{P}^\star(d \mid [s,a])\mathbb{P}(a \mid s)}{\mathbb{P}(d \mid s)} - \frac{\mathbb{P}^\star(d \mid [s,a])\mathbb{P}^\star(a \mid s)}{\mathbb{P}^\star(d \mid s)} \right| \\
&= \sum_{a \in \mathcal{A}} \mathbb{P}^\star(d \mid [s,a]) \frac{|\mathbb{P}(a \mid s)\mathbb{P}^\star(d \mid s) - \mathbb{P}^\star(a \mid s)\mathbb{P}(d \mid s)|}{\mathbb{P}(d \mid s)\mathbb{P}^\star(d \mid s)} \\
&\le \sum_{a \in \mathcal{A}} \mathbb{P}^\star(d \mid [s,a])\mathbb{P}(a \mid s) \frac{|\mathbb{P}^\star(d \mid s) - \mathbb{P}(d \mid s)|}{\mathbb{P}(d \mid s)\mathbb{P}^\star(d \mid s)} + \sum_{a \in \mathcal{A}} \mathbb{P}^\star(d \mid [s,a]) \frac{|\mathbb{P}(a \mid s) - \mathbb{P}^\star(a \mid s)|}{\mathbb{P}^\star(d \mid s)} \\
&= \frac{|\mathbb{P}^\star(d \mid s) - \mathbb{P}(d \mid s)|}{\mathbb{P}^\star(d \mid s)} + \sum_{a \in \mathcal{A}} \mathbb{P}^\star(d \mid [s,a]) \frac{|\mathbb{P}(a \mid s) - \mathbb{P}^\star(a \mid s)|}{\mathbb{P}^\star(d \mid s)} \\
&\le 2 \sum_{a \in \mathcal{A}} \mathbb{P}^\star(d \mid [s,a]) \frac{|\mathbb{P}(a \mid s) - \mathbb{P}^\star(a \mid s)|}{\mathbb{P}^\star(d \mid s)}, \qquad \text{(using the definition of } \mathbb{P}(d \mid s)) \\
&\le \frac{2}{\mathbb{P}^\star(d \mid s)} \|\mathbb{P}(a \mid s) - \mathbb{P}^\star(a \mid s)\|_1.
\end{aligned}
$$

Note that the policy will only be called on a given pair of (s, d) if and only if $\mathbb{P}^\star(d \mid s) > 0$, hence, the constant is bounded. We define $K_s = \max_d \frac{2}{\mathbb{P}^\star(d|s)}$ where maximum is taken over all descriptions $d \in \texttt{supp}\, \mathbb{P}^\star(d \mid s)$. ☐

Theorem 9 (Convergence to Near Optimal Policy)**.** *Fix $t \in \mathbb{N}$, and let $\tilde{\mathbb{P}}_t(a \mid s, d) = \frac{1}{t}\sum_{n=1}^{t} \mathbb{P}_n(a \mid s, d)$ be the average of the agent's policy across epochs. Then for every start state s and description $d \in \texttt{supp}\, \mathbb{P}^\star(d \mid s)$ we have:*

$$\lim_{t \to \infty} (W\tilde{\mathbb{P}}_t)(a \mid s, d) = \mathbb{P}^\star(a \mid s, d).$$

Proof. Let $\tilde{\mathbb{P}}_t(a \mid s) = \sum_d \mathbb{P}^\star(d \mid s)\tilde{\mathbb{P}}_t(a \mid s, d)$. Then it is easy to see that $\tilde{\mathbb{P}}_t(a \mid s) = \frac{1}{t}\sum_{n=1}^{t} \mathbb{P}_n(a \mid s)$. From Theorem 7 we have $\lim_{t\to\infty} \|\tilde{\mathbb{P}}_t(a \mid s) - \mathbb{P}^\star(a \mid s)\|_2 = 0$. As $\mathcal{A}$ is finite dimensional, therefore, $\|\cdot\|_2$ and $\|\cdot\|_1$ are equivalent, i.e., convergence in one also implies convergence in the other. This implies, $\lim_{t\to\infty} \|\tilde{\mathbb{P}}_t(a \mid s) - \mathbb{P}^\star(a \mid s)\|_1 = 0$.

From Lemma 8 we have:

$$\lim_{t \to \infty} \|(W\tilde{\mathbb{P}}_t)(a \mid s, d) - (W\mathbb{P}^\star)(a \mid s, d)\|_1 \le K_s \lim_{t\to\infty} \|\tilde{\mathbb{P}}_t(a \mid s) - \mathbb{P}^\star(a \mid s)\|_1 = 0.$$

This shows $\lim_{t\to\infty}(W\tilde{\mathbb{P}}_t)(a \mid s, d) = (W\mathbb{P}^\star)(a \mid s, d)$. Lastly, we show that the optimal policy $\mathbb{P}^\star(a \mid s, d)$ is a fixed point of W:

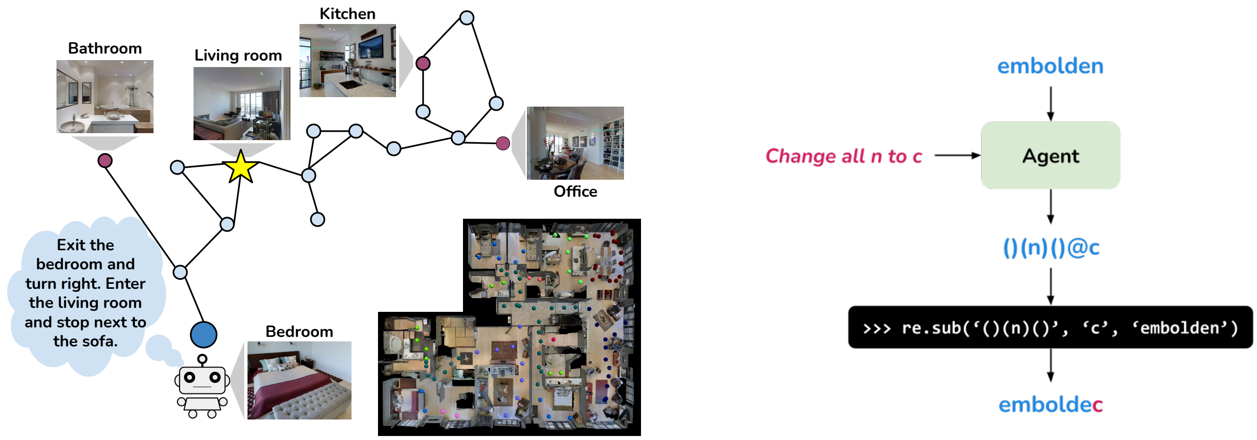
$$(W\mathbb{P}^\star)(a \mid s, d) = \frac{\mathbb{P}^\star(d \mid s, a)\mathbb{P}^\star(a \mid s)}{\sum_{a' \in \mathcal{A}} \mathbb{P}^\star(d \mid s, a')\mathbb{P}^\star(a' \mid s)} = \frac{\mathbb{P}^\star(d, a \mid s)}{\sum_{a' \in \mathcal{A}} \mathbb{P}^\star(d, a' \mid s)} = \frac{\mathbb{P}^\star(d, a \mid s)}{\mathbb{P}^\star(d \mid s)} = \mathbb{P}^\star(a \mid s, d).$$

This completes the proof. ☐

B. Detailed Comparison with Related Works that use Language Description Feedback

Recently, several papers have proposed using language description feedback for speeding up reinforcement learning (Jiang et al., 2019) or aiding exploration of new skills (Colas et al., 2020). While also employing this type of feedback, our work differs from these papers in its (i) research goal, (ii) theoretical study, and (iii) empirical contributions.

Research Goal. The goal of these papers is to use description feedback to create additional training data for a reward-based learning process. In Jiang et al. (2019), the agent has access to a (binary) reward function that it can query. In Colas et al. (2020), the agent does not have access to a reward function, but it is given an exhaustive list of descriptions of each execution. This helps the agent easily construct a reward function and additional examples. In addition, given an execution, the teacher in these papers returns a list of *multiple* descriptions of the execution.



(a) *Vision-language navigation* (NAV): a (robot) agent fulfills a navigational natural-language request in a photo-realistic simulated house. Locations in the house are connected as a graph. In each time step, the agent receives a photo of the panoramic view at its current location (due to space limit, here we only show part of a view). Given the view and the language request, the agent chooses an adjacent location to go to. On average, each house has about 117 locations.

(b) *Word modification via regular expressions* (REGEX): an agent is given an input word and a natural-language request that asks it to modify the word. The agent outputs a regular expression that follows our specific syntax. The regular expression is executed by the Python's `re.sub()` method to generate an output word.

Figure 3. Illustrations of the two request-fulfilling problems that we conduct experiments on.

In contrast, our paper directly investigates the possibility of learning from humans through *pure natural-language communication*. To realistically mimic this scenario, our setup is made more restricted than in these papers: no rewards are given to the agent, and the teacher only gives the agent a single description for each execution.

Having access to a reward function and a list of descriptions adds additional supervision to the language learning problem. Given an execution $\hat{e}$ of a request $d^\star$, suppose a description $\hat{d}$ of $\hat{e}$ is returned along with a reward r. If $r = 1$ (i.e., the execution fulfills the request), the agent can infer that $d^\star$ and $\hat{d}$ both describe the same execution $\hat{e}$, *without* having to ground their meanings to the execution. If $r = 0$, the agent can form a negative example $d^\star \neq \hat{d}$. Similarly, if multiple descriptions is returned, the agent knows that these descriptions all refer to the same execution. Therefore, the language learning problem in these papers are strongly supervised with parallel examples that the agent can exploit to learn a representation of the language space.

In our setup, because the reward is not present and only a single description is given, the agent cannot directly infer a relationship between the description and the request. Only the relationship between the description and the execution can be assumed. Thus *our setup is designed such that language learning relies mostly on grounding to execution*.

Theoretical Study. In these papers, the learning problem is ultimately reduced to reinforcement learning. In contrast, our main algorithm ADEL performs *density estimation* instead of reduction of reinforcement learning. The primary goal of ADEL is to recover the request-conditioned execution distribution. ADEL learns effectively in the face of *uncertainty* of the teacher, which is not modeled by these papers. In addition, we present novel theoretical analysis of the learning dynamic of our algorithm, and gain insights on what conditions the teacher should satisfy to make learning possible.

Empirical Contributions. Last but not least, our paper makes two important empirical contributions:

- We demonstrate that our algorithm can achieve success rates comparable to imitation learning in *photo-realistic environments with rich natural language descriptions*. In contrast, these papers conduct evaluation on toy environments with templated language, and do not compare with imitation learning.
- We present a *scalable data-driven strategy for simulating natural-language description feedback*. In contrast, these papers employ rule-based programs to derive descriptions.

C. Problem settings

Figure 3 illustrates the two problems that we conduct experiments on.

C.1. Vision-Language Navigation

Environment Simulator and Data. We use the Matterport3D simulator and the Room-to-Room dataset[8] developed by Anderson et al. (2018). The simulator photo-realistically emulates the first-person view of a person walking in a house. The dataset contains tuples of human-generated English navigation requests annotated with ground-truth paths in the environments. To evaluate on the test set, the authors require submitting predictions to an evaluation site[9], which limits the number of submissions to five. As our goal is not to establish state-of-the-art results on this task, but to compare performance of multiple learning frameworks, we re-split the data into 4,315 simulation, 2,100 validation, and 2,349 test data points. The simulation split, which is used to simulate the teacher, contains three requests per data point (i.e. $|\mathcal{D}_n| = 3$). The validation and test splits each contains only one request per data point. On average, each request includes 2.5 sentences and 26 words. The word vocabulary size is 904 and the average number of optimal actions required to reach the goal is 6.

Simulated Teacher. We use SDTW (Magalhaes et al., 2019) as the `perf` metric and set the threshold $\tau = 0.5$. The SDTW metric re-weights success rate by the shortest (order-preserving) alignment distance between the predicted and the ground-truth paths, thus capturing the fidelity of the agent execution to the intent of the input request.

Approximate marginal $P_{\pi_\omega}(e \mid s_1)$. The approximate marginal is a function that takes in a start location s_1 and randomly samples a shortest path on the environment graph that starts from s_1 and has (unweighted) length between 2 and 6.

C.2. Word Modification via Regular Expressions

Regular Expression Compiler. We use Python 3.7's `re.sub(pattern, replace, string)` method as the regular expression compiler. The method replaces every substring of `string` that matches a regular expression `pattern` with the string `replace`. A regular expression predicted by our agent $\hat{a}_{1:H}$ has the form "`pattern@replace`", where `pattern` and `replace` are strings and `@` is the at-sign character. For example, given the word *embolden* and the request "*replace all n with c*", the agent should ideally generate the regular expression "()(n)()@c". We then split the regular expression by the character `@` into a string `pattern` = "*()(n)()*" and a string `replace` = "*c*". We execute the command `re.sub('()(n)()', 'c', 'embolden')` to obtain the output word *emboldec*.

Data. We use the data collected by Andreas et al. (2018). The authors presented crowd-workers with pairs of input and output words where the output words are generated by applying regular expressions onto the input words. Workers are asked to write English requests that describe the change from the input words to the output words. In the end, the authors extracted 1,917 request templates from the human-generated requests. Each request template is annotated with a regular expression template that it describes. For example, a template has the form *add an `AFTER` to the start of words beginning with `BEFORE`*, where `AFTER` and `BEFORE` can be replaced with latin characters to form a request. Since the original dataset is not designed to evaluate generalization to previously unseen request templates, we modified the script provided by the authors to generate a new dataset where the simulation and evaluation requests are generated from disjoint sets of request templates. We select 110 regular expressions templates that are each annotated with more than one request template. Then, we further remove pairs of regular expression and request templates that are mistakenly paired. We end up with 1111 request templates describing these 110 regular expression templates. We use these templates to generate pairs of requests and regular expressions. In the end, our dataset consists of 114,503 simulation, 6,429 validation, and 6,429 test data points. The sets of simulation, validation, and test request templates are disjoint.

Simulated Teacher. We extend the performance metric in §5.1 `perf` to evaluating multiple executions. Concretely, given executions $\{w_j^{\text{inp}}, \hat{w}_j^{\text{out}}\}_{j=1}^{K}$, the metric counts how many pairs where the predicted output word matches the ground-truth: $\sum_{j=1}^{K} \mathbb{1}\left\{\hat{w}_j^{\text{out}} = w_j^{\text{out}}\right\}$. We set the threshold $\tau = K$.

[8] https://github.com/peteanderson80/Matterport3DSimulator/blob/master/tasks/R2R/data/download.sh

[9] https://eval.ai/web/challenges/challenge-page/97/overview

Algorithm 5 Interactive learning from activity descriptions (experimental version).

1: **Input**: teacher model $\mathbb{P}_T(d \mid e)$, approximate marginal $\mathbb{P}_{\pi_\omega}(e \mid s_1)$, mixing rate $\lambda \in [0, 1]$
2: Initialize agent policy $\pi_\theta : \mathcal{S} \times \mathcal{D} \to \Delta(\mathcal{A})$
3: Initialize agent policy $\pi_\beta : \mathcal{S} \times \mathcal{D} \to \Delta(\mathcal{A})$
4: **for** $n = 1, 2, \cdots, N$ **do**
5: Word samples $q = (R, d^\star, s_1) \sim \mathbb{P}^\star(\cdot)$
6: Agent generates $\hat{e} \sim \mathbb{P}_{\pi_\beta}(\cdot \mid s_1, d^\star)$
7: Teacher generates description $\hat{d} \sim \mathbb{P}_T(\cdot \mid \hat{e})$
8: Agent samples $\tilde{e} \sim \mathbb{P}_{\pi_\omega}(\cdot \mid s_1)$
9: Compute losses:

$$\mathcal{L}(\theta) = \sum_{(s, \hat{a}_s) \in \hat{e}} \log \pi_\theta \left(\hat{a}_s \mid s, \hat{d} \right)$$

$$\mathcal{L}(\beta) = \lambda \sum_{(s, \tilde{a}_s) \in \tilde{e}} \log \pi_\beta \left(a_s \mid s, \hat{d} \right) + (1 - \lambda) \sum_{(s, \tilde{a}_s) \in \hat{e}} \log \pi_\beta \left(\tilde{a}_s \mid s, \hat{d} \right)$$

10: Compute gradients $\nabla \mathcal{L}(\theta)$ and $\nabla \mathcal{L}(\beta)$
11: Use gradient descent to update θ and β with $\nabla \mathcal{L}(\theta)$ and $\nabla \mathcal{L}(\beta)$, respectively
return $\pi : s, d \mapsto \text{argmax}_a \, \pi_\theta(a \mid s, d)$

Approximate marginal $P_{\pi_\omega}(e \mid s_1)$. The approximate marginal is a uniform distribution over a dataset of (unlabeled) regular expressions. Each regular expression is generated using the template[10] used by Andreas et al. (2018) to construct their dataset.

D. Practical Implementation of ADEL

In our experiments, we employ the following implementation of ADEL (Alg 5), which learns a policy π_β such that $\mathbb{P}_{\pi_\beta}(e \mid s_1, d)$ approximates the mixture $\tilde{\mathbb{P}}(e \mid s_1, d)$ in Alg 3. In each episode, we sample an execution $\hat{e}$ using the policy π_β. Then, similar to Alg 3, we ask the teacher $\mathbb{P}_T$ for a description of $\hat{e}$ and the use the pair $\left(\hat{e}, \hat{d}\right)$ to update the agent policy π_θ. To ensure that $\mathbb{P}_{\pi_\beta}$ approximates $\tilde{\mathbb{P}}$, we draw a sample $\tilde{e}$ from the approximate marginal $\mathbb{P}_{\pi_\omega}(e \mid s_1)$ and update π_β using a λ-weighted loss of the log-likelihoods of the two data points $\left(\tilde{e}, \hat{d}\right)$ and $\left(\hat{e}, \hat{d}\right)$. We only use $\left(\hat{e}, \hat{d}\right)$ to update the agent policy π_θ.

An alternative (naive) implementation of sampling from the mixture $\tilde{\mathbb{P}}$ is to first choose a policy between π_ω (with probability λ) and π_θ (with probability $1 - \lambda$), and then use this policy to generate an execution. Compared to this approach, our implementation has two advantages:

1. Sampling from the mixture is simpler: instead of choosing between π_θ and π_ω, we always use π_β to generate executions;

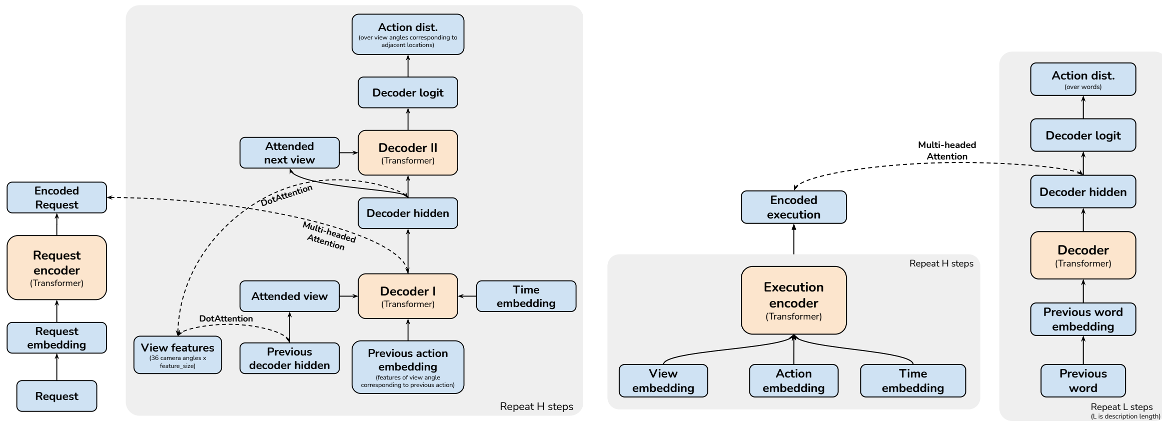
2. More importantly, samples are more diverse: in the naive approach, the samples are either completely request-agnostic (if generated by π_ω) or completely request-guided (if generated by π_θ). As a machine learning-based model that learns from a mixture of data generated by π_ω and π_θ, π_β can generalize and generate executions that are partially request-agnostic.

E. Training details

Reinforcement learning's continuous reward. In REGEX, the continuous reward function is

$$\frac{|w^{\text{out}}| - \texttt{editdistance}\left(\hat{w}^{\text{out}}, w^{\text{out}}\right)}{|w^{\text{out}}|} \tag{16}$$

[10] https://github.com/jacobandreas/l3/blob/master/data/re2/generate.py



(a) Student model (b) Teacher model

Figure 4. Student and teacher models in NAV.

Hyperparameter	Vision-Language Navigation	Word Modification
Student policy π_θ and **Teacher's describer model** $\tilde{P}_T$		
Base architecture	LSTM	Transformer
Hidden size	256	512
Number of hidden layers (of each encoder or decoder)	1	1
Request word embedding size	256	128
Character embedding size	-	8
Time embedding size	256	-
Attention heads	8	1
Input-view feature size	2048	-
Teacher simulation		
`perf` metric	STDW (Magalhaes et al., 2019)	Number of output words matching ground-truths
Number of samples for approximate pragmatic inference ($\lvert\mathcal{D}_{\text{cand}}\rvert$)	5	10
Threshold (τ)	0.5	10
Training		
Time horizon (H)	10	40
Batch size	32	32
Learning rate	10^{-4}	10^{-3}
Optimizer	Adam	Adam
Number of training iterations	25K	30K

Table 5. Hyperparameters for training with the ADEL algorithm.

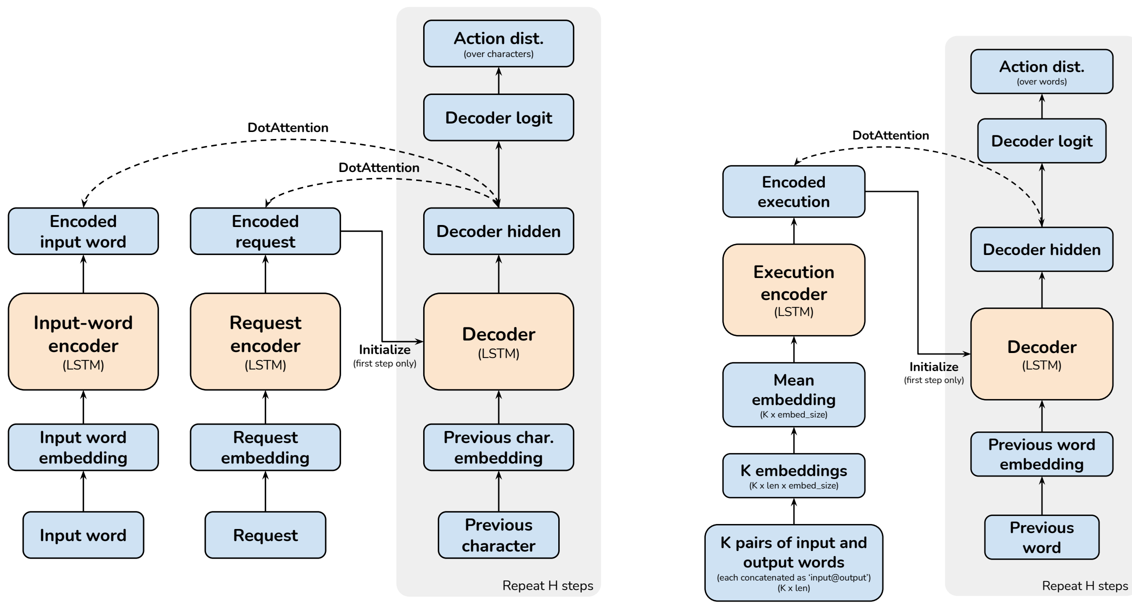
where w^{out} is the ground-truth output word, $\hat{w}^{\text{out}}$ a distance metric, $\texttt{editdistance}(.,.)$ is the string edit distance computed by the Python's editdistance module. In NAV, the continuous reward function is

$$\frac{\texttt{shortest}\,(s_1, s_g) - \texttt{shortest}\,(s_H, s_g)}{\texttt{shortest}\,(s_1, s_g)} \tag{17}$$

where s_1 is the start location, s_g is the goal location, s_H is the agent's final location, and $\texttt{shortest}(.,.)$ is the shortest-path distance between two locations (according to the environment's navigation graph).

Model architecture. Figure 4 and Figure 5 illustrate the architectures of the models that we train in two problems, respectively. For each problem, we describe the architectures of the student policy π_θ and the teacher's language model $\tilde{\mathbb{P}}(d \mid e)$. All models are encoder-decoder models but the NAV models use LSTM as the recurrent module while REGEX models use Transformer.

Hyperparameters. Model and training hyperparameters are provided in Table 5. Each model is trained on a single NVIDIA V100 GPU. Training with the ADEL algorithm takes about 22 hours for NAV and 17 hours for REGEX.



(a) Student model

(b) Teacher model

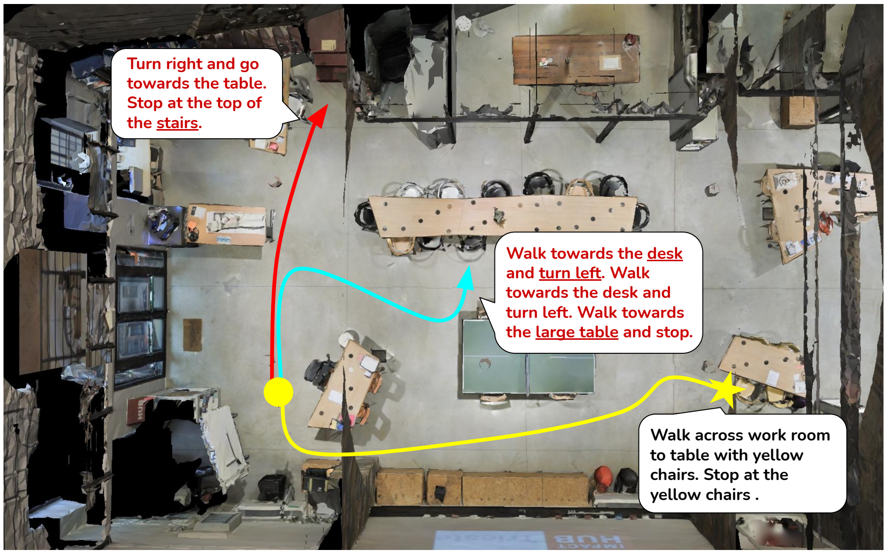
Figure 5. Student and teacher models in REGEX.

Input word	Output word	Teacher descriptions
attendant	xjtendxjt	replace [a] and the letter that follows it with an [x j]
disclaims	esclaims	if the word does not begin with a vowel , replace the first two letters with [e]
inculpating	incuxlpating	for any instance of [l] add a [x] before the [l]
flanneling	glanneling	change the first letter of the word to [g]
dhoti	jhoti	replaced beginning of word with [j]
stuccoing	ostuccoing	all words get a letter [o] put in front
reappearances	reappearanced	if the word ends with a consonant , change the consonant to [d]
bigots	vyivyovyvy	replace each consonant with a [v y]

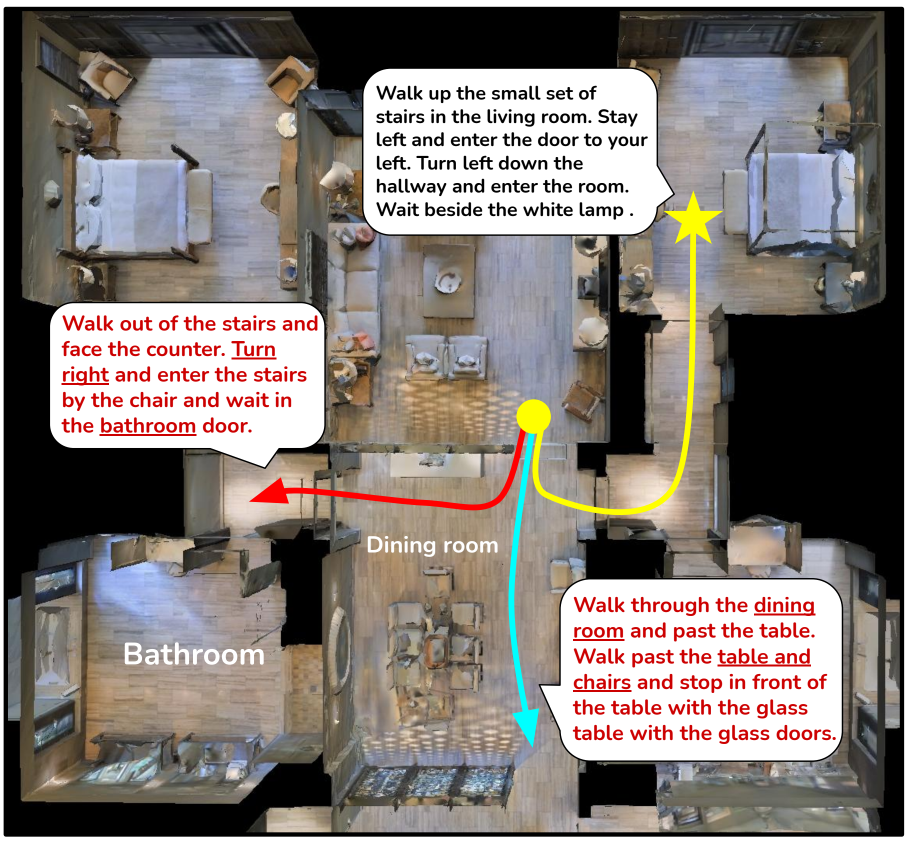
Table 6. Qualitative examples in the REGEX problem. We show pairs of input and output words and how the teacher's language model $\tilde{\mathbb{P}}(d \mid e)$ describes the modifications applied to the input words.

F. Qualitative examples

Figure 6 and Table 6 show the qualitative examples in the NAV and REGEX problems, respectively.



(a)



(b)

Figure 6. Qualitative examples in the NAV problem. The **black texts** are the initial requests $d^{\star}$ generated by humans. The paths are the ground-truth paths implied by the requests. The and paths are taken by the agent to explore the environments. Here, we only show two explorative paths per example. The **red texts** are the descriptions $\hat{d}$ generated by the teacher's learned (conditional) language model $\tilde{\mathbb{P}}(d \mid e)$. We show the bird-eye views of the environments for better visualization; the agent only has access to the first-person panoramic views at its locations.