Executing Instructions in Situated Collaborative Interactions

Alane Suhr
Cornell University
suhr@cs.cornell.edu

Claudia Yan
IBM
claudiab.yan@gmail.com

Jacob Schluger[*]
Cornell University
jes543@cornell.edu

Stanley Yu[*]
Columbia University
stanley.yu@columbia.edu

Hadi Khader[**]
Intel
hadi.kh.khader@gmail.com

Marwa Mouallem[**]
IBM
marwamouallem@gmail.com

Iris Zhang
Facebook
irisz@fb.com

Yoav Artzi
Cornell University
yoav@cs.cornell.edu

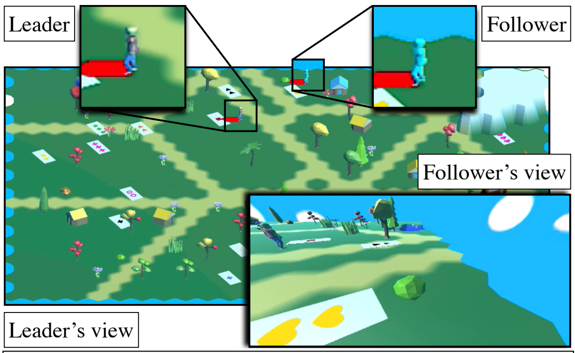
Abstract

We study a collaborative scenario where a user not only instructs a system to complete tasks, but also acts alongside it. This allows the user to adapt to the system abilities by changing their language or deciding to simply accomplish some tasks themselves, and requires the system to effectively recover from errors as the user strategically assigns it new goals. We build a game environment to study this scenario, and learn to map user instructions to system actions. We introduce a learning approach focused on recovery from cascading errors between instructions, and modeling methods to explicitly reason about instructions with multiple goals. We evaluate with a new evaluation protocol using recorded interactions and online games with human users, and observe how users adapt to the system abilities.

1 Introduction

Sequential instruction scenarios commonly assume only the system performs actions, and therefore only its behavior influences the world state. This ignores the collaborative potential of such interactive scenarios and the challenges it introduces. When the user acts in the world as well, they can adapt to the system abilities not only by adopting simpler language, but also by deciding to accomplish tasks themselves. The system must then recover from errors as new instructions arrive and be robust to changes in the environment that are not a result of its own actions.

In this paper, we introduce CEREALBAR, a collaborative game with natural language instruction, and design modeling, learning, and evaluation methods for the problem of sequential instruction following in collaborative interactions. In CEREALBAR, two agents, a leader and a follower,



...
$\bar{x}_3$: *turn left and head toward the yellow hearts, but don't pick them up yet. I'll get the next card first.*
$\bar{x}_4$: ***Okay, pick up yellow hearts and run past me toward the bush sticking out, on the opposite side is 3 green stars***
`[Set made.  New score:  4]`
...

Figure 1: A snapshot from an interaction in CEREALBAR. The current instruction is in bold. The large image shows the entire environment. This overhead view is available only to the leader. The follower sees a first-person view only (bottom right). The zoom boxes (top) show the leader and follower.

move in a 3D environment and collect valid sets of cards to earn points. A valid set is a set of three cards with distinct color, shape, and count. The game is turn-based, and only one player can act in each turn. In addition to collecting cards, the leader sends natural language instructions to the follower. The follower's role is to execute these instructions. Figure 1 shows a snapshot from the game where the leader plans to pick up a nearby card (red square) and delegates to the follower two cards, one close and the other much further away. Before that, the leader planned ahead and asked the follower to move in preparation for the next set. The agents have different skills to incentivize collaboration. The follower has more moves per turn, but can only see from first-person view, while the leader observes the entire environment but has fewer moves. This makes natural language inter-

[*],[**]: Equal contribution. All work done at Cornell.

action key to success. We address the problem of mapping the leader instructions to follower actions. In addition to the collaborative challenges, this requires grounding natural language to resolve spatial relations and references to objects, reason about dependencies on the interaction history, react to the changing environment as cards appear and disappear, and generate actions.

CEREALBAR requires reasoning about the changing environment (e.g., when selecting cards) and instructions with multiple goals (e.g., selecting multiple cards). We build on the Visitation Prediction Network model (VPN; Blukis et al., 2018b), which casts planning as mapping instructions to the probability of visiting positions in the environment. Our new model generalizes the planning space of VPN to reason about intermediate goals and obstacles, and includes recurrent action generation for trajectories with multiple goals.

We collect 1,202 human-to-human games for training and evaluation. While our model could be trained from these recorded games only, it would often fail when an instruction would start at the wrong position because of an error in following the previous one. We design a learning algorithm that dynamically augments the data with examples that require recovering from such errors, and train our model to distinguish such recovery reasoning from regular instruction execution.

Evaluation with recorded games poses additional challenges. As agent errors lead to unexpected states, later instructions become invalid. Because measuring task completion from such states is meaningless, we propose *cascaded evaluation*, a new evaluation protocol that starts the agent at different points in the interaction and measures how much of the remaining instructions it can complete. In contrast to executing complete sequences or single instructions, this method allows to evaluate all instructions while still measuring the effects of error propagation.

We evaluate using both static recorded games and live interaction with human players. Our human evaluation shows users adapt to the system and use the agent effectively, scoring on average 6.2 points, compared to 12.7 for human players. Our data, code, and demo videos are available at lil.nlp.cornell.edu/cerealbar/.

2 Setup and Technical Overview

We consider a setup where two agents, a leader and a follower, collaborate. Both execute actions in a shared environment. The leader, additionally, instructs the follower using natural language. The leader goal is to maximize the task reward, and the follower goal is to execute leader instructions. We consider a turn-based version, where at each turn only one agent acts. We instantiate this scenario in CEREALBAR, a navigation card game (Figure 1), where a leader and follower move in an environment selecting cards to complete sets.[1]

CEREALBAR Overview The objective of CEREALBAR is to earn points by selecting valid sets of cards. A valid set has three cards with distinct color, shape, and count. When the only cards selected in the world form a valid set, the players receive a point, the selected cards disappear, three new cards are added randomly, and the number of remaining turns increases. The increase in turns decays for each set completed. An agent stepping on a card flips its selection status. The players form sets together. The follower has more steps per turn than the leader. This makes using the follower critical for success. The follower only sees a first-person view of the environment, preventing them from planning themselves, and requiring instructions to be sensible from the follower's perspective. The leader chooses the next target set, plans which of the two players should get which card, and instructs the follower. The follower can not respond to the leader, and should not plan themselves, or risk sabotaging the leader's plan, wasting moves and lowering their potential score. Followers mark an instruction as finished before observing the next one. This provides alignment between instructions and follower actions. In contrast to the original setup that we use for data collection, in our model (Section 4), we assume the follower has full observability, leaving the challenge of partial observability for future work. Appendix A provides further game design details.

Problem Setup We distinguish between the world state and the interaction state. Let $\mathcal{S}$ be the set of all world states, Γ be the set of all interaction states, and $\mathcal{X}$ be the set of all natural language instructions. A world state $s \in \mathcal{S}$ describes the current environment. In CEREALBAR, the world state describes the spatial environment, the location of cards, whether they are selected or not, and the location of the agents. An interaction state $\gamma \in \Gamma$ is a tuple $\langle \bar{Q}, \alpha, \psi \rangle$. The first-in-first-out

[1]The name CEREALBAR does not carry special meaning. It was given to the project early on, and we came to like it. Our game is inspired by the card game Set.

queue $\bar{Q} = [\bar{x}_q, \ldots, \bar{x}_{q'}]$ contains the instructions $\bar{x}_i \in \mathcal{X}$ available to execute. The current instruction is the left-most instruction $\bar{x}_q$. The current turn-taker $\alpha \in \{\text{Leader}, \text{Follower}\}$ indicates the agent currently executing actions, and $\psi \in \mathbb{N}_{\geq 0}$ is the number of steps remaining in the current turn.

At each time step, the current turn-taker agent takes an action. An action may be the leader issuing an instruction, or either agent performing an action in the environment. Let $\mathcal{A} = \mathcal{A}_w \cup \{\texttt{DONE}\} \cup \mathcal{X}$ be the set of all actions. The set $\mathcal{A}_w$ includes the actions available to the agents in the environment. In CEREALBAR, this includes moving forward or backward, and turning left or right. Moving onto a card flips it selection status. DONE indicates completing the current instruction for the follower or ending the turn for the leader. An instruction action $a = \bar{x} \in \mathcal{X}$ can only be taken by the leader and adds the instruction $\bar{x}$ to the queue $\bar{Q}$. The effect of each action is determined by the transition function $\mathcal{T} : \mathcal{S} \times \Gamma \times \mathcal{A} \rightarrow \mathcal{S} \times \Gamma$, which is formally defined in Appendix B. Only world actions $a \in \mathcal{A}_w$ decrease the remaining steps ψ.

The goal of the leader is to maximize the total reward of the interaction. An interaction $\bar{I} = \langle (s_1, \gamma_1, a_1), \ldots, (s_{|\bar{I}|}, \gamma_{|\bar{I}|}, a_{|\bar{I}|}) \rangle$ is a sequence of state-action tuples, where $\mathcal{T}(s_i, \gamma_i, a_i) = (s_{i+1}, \gamma_{i+1})$. The reward function $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ assigns a numerical reward to a world state and an action. The total reward of an interaction $\bar{I}$ is $\sum_{i=0}^{|\bar{I}|} R(s_i, a_i)$. In CEREALBAR, the agents receive a reward of 1 when a valid set is selected.

Task Our goal is to learn a follower policy to execute the leader instructions. At time t, given the current world and interaction states s_t and γ_t, and the interaction so far $\bar{I}_{<t}$, the follower policy $\pi(s_t, \gamma_t, \bar{I}_{<t})$ predicts the next action a_t.

Model We decompose the follower policy $\pi(s_t, \gamma_t, \bar{I}_{<t})$ to predicting a set of distributions over positions in the environment, including positions to visit, intermediate goals (e.g., cards to select), positions to avoid (e.g., cards not to touch), and positions that are not passable. These distribution are used in a second stage to generate a sequence of actions. Section 4 describes the model.

Learning We assume access to a set of N recorded interactions $\{\bar{I}^{(i)}\}_{i=1}^{N}$, and create examples where each instruction is paired with a sequence of state-action tuples. We maximize the action-level cross entropy objective, and use two auxiliary objectives (Section 5). We first train each stage of the model separately, and then fine-tune them jointly. During fine-turning, we continuously generate additional examples using model failures. These examples help the agent to learn how to recover from errors in prior instructions.

Evaluation We measure correct execution of instructions and the overall game reward. We assume access to a test set of M recorded interactions $\{\bar{I}^{(i)}\}_{i=1}^{M}$. We measure instruction-level and interaction-level performance, and develop *cascaded evaluation*, an evaluation protocol that provides a more graded measure than treating each interaction as a single example, while still accounting for error propagation (Section 6). Finally, we conduct online evaluation with human leaders.

3 Related Work

Goal-driven natural language interactions have been studied in various scenarios, including dialogue where only one side acts in the world (Anderson et al., 1991; Williams et al., 2013; Vlachos and Clark, 2014; de Vries et al., 2018; Kim et al., 2019; Hu et al., 2019), coordination for agreed selection of an object (He et al., 2017; Udagawa and Aizawa, 2019), and negotiation (Lewis et al., 2017; He et al., 2018). We focus on collaborative interactions where both the user and the system perform sequences of actions in the same environment. This allows the user to adapt to the language understanding ability of the system and balance between delegating goals to it and accomplishing them themselves. For example, a user may decide to complete a short but hard-to-describe task and delegate to the system a long but easy-to-describe one. In prior work, in contrast, recovery is limited to users paraphrasing their requests. The Cards corpus (Djalali et al., 2011, 2012; Potts, 2012) was used for linguistic analysis of collaborative bi-directional language interaction. The structure of collaborative interactions was also studied using Wizard-of-Oz studies (Lochbaum, 1998; Sidner et al., 2000; Koulouri and Lauria, 2009). In contrast, we focus on building agents that follow instructions. Ilinykh et al. (2019) present a corpus for the related task of natural language coordination in navigation. Collaboration has also been studied for emergent communication (e.g., Andreas et al., 2017; Evtimova et al., 2017).

Understanding sequences of natural language utterances has been addressed using semantic parsing (e.g., Miller et al., 1996; MacMahon et al., 2006; Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013; Artzi et al., 2014; Long et al., 2016;

Iyyer et al., 2017; Suhr et al., 2018; Arkin et al., 2017; Broad et al., 2017). Interactions were also used for semantic parser induction (Artzi and Zettlemoyer, 2011; Thomason et al., 2015; Wang et al., 2016). These methods require hand-crafted symbolic meaning representation, while we use low-level actions (Suhr and Artzi, 2018). The interactions in our environment interleave actions of both agents with leader utterances, an aspect not addressed by these methods. Executing single instructions has been widely studied (e.g., Tellex et al., 2011; Duvallet et al., 2013; Misra et al., 2017, 2018; Anderson et al., 2018; Blukis et al., 2018a,b; Chen et al., 2019). The distinction we make between actions specified in the instruction and implicit recovery actions is similar to how Artzi and Zettlemoyer (2013) use implicit actions for single instructions. Finally, our model is based on the VPN model of Blukis et al. (2018b). While we assume full observability, their original work did not. This indicates that our model is likely to generalize well to partially observable scenarios.

4 Model

We use a two-stage model for the follower policy $\pi(s_t, \gamma_t, \bar{I}_{<t})$, where s_t is a world state, γ_t is an interaction state, and $\bar{I}_{<t}$ is the interaction history. The instruction $\bar{x}$ that is the first in the queue $\bar{Q}_t$, which is part of γ_t, is the currently executed instruction. In our model, we assume the follower observes the entire environment. First, we map $\bar{x}$ and s_t to distributions over locations in the environment, including what locations to visit and what are the goals. These distributions are considered as an execution plan, and are used to generate a sequence of actions in the second stage. The distribution can also be used to easily easily visualize the agent plan. The first stage is used when starting a new instruction, and the predicted distributions are re-used for all actions for that instruction. Figure 2 illustrates the architecture and the distributions visualization. The two-stage approach was introduced by Blukis et al. (2018b). We generalize its planning space and add a recurrent action generator for execution.

Input Representation The inputs to the first stage are the instruction $\bar{x}$ and the world state s_t. We generate feature maps for both. We use a learned embedding function $\phi^{\mathcal{X}}$ and a bi-directional recurrent neural network (RNN; Elman, 1990) with a long short-term memory cell (LSTM; Hochreiter and Schmidhuber, 1997) RNN$^{\mathcal{X}}$ to map $\bar{x}$ to a vector $\bar{\mathbf{x}}$. The world state s_t is a 3D tensor that encodes the properties of each position. The dimensions of s_t are $P \times W \times H$, where P is the number of properties, and W and H are the environment width and height. Each of the $W \times H$ positions is represented in s_t as a binary vector of length P. For example, a position with a red hut will have 1's for the *red* and *hut* dimensions and 0's for all other dimensions. We map the world state to a tensor feature map $\mathbf{F}_0$ by embedding s_t and processing it using the text representation $\bar{\mathbf{x}}$. We use a learned embedding function $\phi^{\mathcal{S}}$ to map each position vector to a dense embedding of size N_s by summing embeddings of each of the position's properties. The embeddings are combined to a tensor $\mathbf{S}$ of dimension $N_s \times W \times H$ representing a featurized global view of the environment. We create a text-conditioned state representation by creating a kernel $\mathbf{K}_s$ and convolving with it over $\mathbf{S}$. We use a linear transformation to create $\mathbf{K}_s = \mathbf{W}_s \bar{\mathbf{x}} + \mathbf{b}_s$, where $\mathbf{W}_s$ and $\mathbf{b}_s$ are learned weights. We reshape $\mathbf{K}_s$ to a 1×1 convolution kernel with $N_{s'}$ output channels, and compute $\mathbf{S}' = \mathbf{S} * \mathbf{K}_s$. We concatenate $\mathbf{S}$ and $\mathbf{S}'$ along the channel dimension and rotate and center so the follower position is at center pixel to generate $\mathbf{F}_0$.[2]

Stage 1: Plan Prediction We treat plan generation as predicting distributions over positions ρ in the environment. There are $W \times H$ possible positions. We predict four distributions: (a) $p(\rho \mid s_t, \bar{x})$, the probability of visiting ρ while executing the instruction $\bar{x}$; (b) $p(\text{GOAL} = 1 \mid \rho, s_t, \bar{x})$, the binary probability that ρ is a goal (i.e., GOAL $= 1$ when containing a card to select); (c) $p(\text{AVOID} = 1 \mid \rho, s_t, \bar{x})$, the binary probability that the agent must not pass in ρ (i.e., AVOID $= 1$ when it contains a card that should not be touched); and (d) $p(\text{NOPASS} = 1 \mid \rho, s_t, \bar{x})$, the binary probability the agent cannot pass in ρ (i.e., NOPASS $= 1$ when it contains another object).

We use LingUNet (Misra et al., 2018) to predict the distributions. The inputs to LingUNet are the instruction embedding $\bar{\mathbf{x}}$ and featurized world state $\mathbf{F}_0$, which is relative to the agent's frame of reference. The output are four matrices, each of dimension $W \times H$ corresponding to the environment. LingUNet is formally defined in Misra et al. (2018) and Appendix D. Roughly

[2]Appendix D describes the relationship between the environment representations and the agent's initial and current orientation.

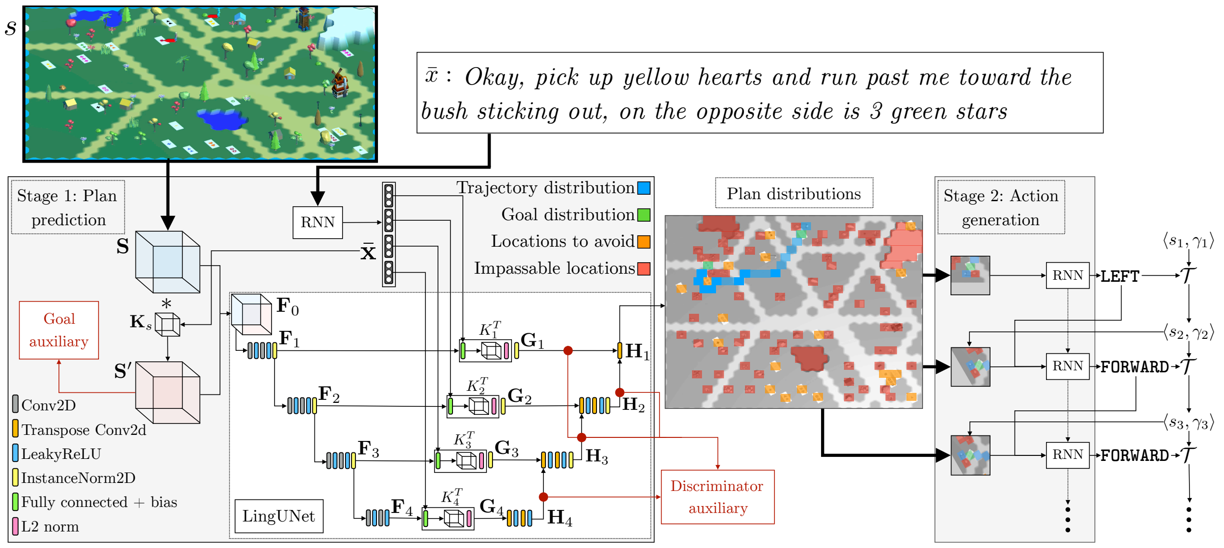


Figure 2: Illustration of the model architecture. Given the instruction $\bar{x}$ and the world state s, we compute $\mathbf{F}_0$ from the embeddings of the instruction $\bar{\mathbf{x}}$ and environment $\mathbf{S}$. We use LINGUNET to predict four distributions, which are visualized over the map (grayscaled to emphasize the distributions). We show three action generation steps. Each step receives the map cropped around the agent and the previous action, and outputs the next action.

speaking, LINGUNET reasons about the environment representation $\mathbf{F}_0$ at L levels. First, $\mathbf{F}_0$ is used to generate feature maps of decreasing size $\mathbf{F}_j$, $j = 1 \dots L$ using a series of convolutions. We create convolution kernels from the instruction representation $\bar{\mathbf{x}}$, and apply them to the feature maps $\mathbf{F}_j$ to generate text-conditioned feature maps $\mathbf{G}_j$. Finally, feature maps of increasing size $\mathbf{H}_j$ are generated using a series of L deconvolutions. The last deconvolution generates a tensor of size $4 \times W \times H$ with a channel for each of the four distributions. We use a softmax over one channel to compute $p(\rho \mid s_t, \bar{x})$. Because the other distributions are binary, we use a sigmoid on each value independently for the other channels. When computing $p(\text{GOAL} = 1 \mid \rho, s_t, \bar{x})$ and $p(\text{AVOID} = 1 \mid \rho, s_t, \bar{x})$ we mask positions without objects that can be changed (i.e., positions without cards) to assign them zero probability.

Stage 2: Action Generation We use the four distributions to generate a sequence of actions. We concatenate the distributions channel-wise to a tensor $\mathbf{P} \in \mathbb{R}^{4 \times W \times H}$. We use a forward LSTM RNN to predict a sequence of actions. At each prediction step t, we rotate, transform, and crop $\mathbf{P}$ to generate the egocentric tensor $\mathbf{P}'_t \in \mathbb{R}^{N' \times C \times C}$, where the agent is always at the center and facing in the same direction, such that $\mathbf{P}'_t$ is relative to the agent's current frame of reference. The input to the action generation RNN at time t is:

$$\begin{aligned}\mathbf{p}'_t &= \text{vec}(\text{NORM}(\text{RELU}(\text{CNN}^P(\mathbf{P}'_t))))\\ \mathbf{p}_t &= \text{RELU}(\mathbf{W}_2^P[\mathbf{p}'_t; \text{RELU}(\mathbf{W}_1^P \mathbf{p}'_t + \mathbf{b}_1^P)] + \mathbf{b}_2^P)\ ,\end{aligned}$$

where CNN^P is a convolutional layer, RELU is a non-linearity, NORM is instance normalization (Ulyanov et al., 2017), and $\mathbf{W}_1^P$, $\mathbf{W}_2^P$, $\mathbf{b}_1^P$, $\mathbf{b}_2^P$ are learned weights. The action probability is:

$$\begin{aligned}\mathbf{h}_t &= \text{RNN}^{\mathcal{A}}(\mathbf{h}_{t-1}, [\phi^{\mathcal{A}}(a_{t-1}); \mathbf{p}_t])\\ p(a) &\propto \exp(\mathbf{W}^{\mathcal{A}}[\mathbf{h}_t; \mathbf{p}_t] + \mathbf{b}^{\mathcal{A}})\ , \qquad (1)\end{aligned}$$

where $\text{RNN}^{\mathcal{A}}$ is an LSTM RNN, $\phi^{\mathcal{A}}$ is a learned action embedding function, a_0 is a special `START` action, and $\mathbf{W}^{\mathcal{A}}$ and $\mathbf{b}^{\mathcal{A}}$ are learned. During inference, we assign zero probabilities to actions a when $\mathcal{T}_w(s_t, a)$ is invalid (Appendix B), for example when an agent would move into an obstacle.

5 Learning

We assume access to a set of N recorded interactions $\{\bar{I}^{(i)}\}_{i=1}^N$. We generate instruction-level examples $\mathcal{D} = \bigcup_{i=1}^N \{\bar{I}^{(i,j)}\}_{j=1}^{M^{(i)}}$, where $M^{(i)}$ is the number of examples from $\bar{I}^{(i)}$. Each $\bar{I}^{(i,j)} = \langle (s_1^{(i,j)}, \gamma_1^{(i,j)}, a_1^{(i,j)}), \dots, (s_k^{(i,j)}, \gamma_k^{(i,j)}, a_k^{(i,j)}) \rangle$ is a subsequence of tuples in $\bar{I}^{(i)}$, where $a_1^{(i,j)}$ is the first action the follower takes after observing the j-th instruction in $\bar{I}^{(i)}$, and $a_k^{(i,j)}$ is the `DONE` action completing that instruction. We first estimate the parameters for plan prediction θ_1 and action generation θ_2 separately (Section 5.1), and then fine-tune jointly with data augmentation (Section 5.2).

5.1 Pretraining

Stage 1: Plan Prediction The input of Stage 1 is the world state s_1 and the instruction $\bar{x}$ at the head of the queue $\bar{Q}$.[3] We generate labels for the four

[3]We omit example indices for succinctness.

output distributions using $\bar{I}^{(i,j)}$. The visitation distribution $p(\rho \mid s_1, \bar{x})$ label is proportional to number of states $s_t \in \bar{I}^{(i,j)}$ where the follower is in position ρ. The goal and avoidance distributions model how the agent plans to manipulate parts of its environment to achieve the specified goals, but avoid manipulating other parts. In CEREALBAR, this translates to changing the status of cards, or avoiding doing so. For $p(\text{GOAL} = 1 \mid \rho, s_1, \bar{x})$, we set the label to 1 for all ρ that contain a card that the follower changed its selection status in $\bar{I}^{(i,j)}$, and 0 for all other positions. Similarly, for the avoidance distribution $p(\text{AVOID} = 1 \mid \rho, s_1, \bar{x})$, the label is 1 for all ρ that have cards that the follower does not change during the interaction $\bar{I}^{(i,j)}$. Finally, for $p(\text{NOPASS} = 1 \mid \rho, s_1, \bar{x})$, the label is 1 for all positions the agent cannot move onto, and zero otherwise. We define four cross-entropy losses: visitation $\mathcal{L}_V$, goal $\mathcal{L}_G$, avoidance $\mathcal{L}_A$, and no passing $\mathcal{L}_P$. We also use an auxiliary cross-entropy goal-prediction loss $\mathcal{L}_{G'}$ using a probability $p'_G(\text{GOAL} = 1 \mid \rho, s_1, \bar{x})$ we predict from the pre-LINGUNET representation $\mathbf{S}'$ by classifying each position. The complete loss is a weighted sum with coefficients:[4]

$$\begin{aligned}\mathcal{L}_1(\theta_1) =& \lambda_V \mathcal{L}_V(\theta_1) + \lambda_G \mathcal{L}_G(\theta_1) + \lambda_A \mathcal{L}_A(\theta_1) \\ &+ \lambda_P \mathcal{L}_P(\theta_1) + \lambda_{G'} \mathcal{L}_{G'}(\theta_1) \ .\end{aligned}$$

Stage 2: Action Generation We use the gold distribution to create the input $\mathbf{P}$, and optimize towards the annotated set of actions using teacher forcing (Williams and Zipser, 1989). We compute the loss only over actions taken by the follower:

$$\mathcal{L}_2(\theta_2) = -\sum_{t=1}^{n} \mathbb{1}_{\alpha_t = \text{Follower}} p(a_t) \ ,$$

where $p(a_t)$ is computed by Equation 1.

5.2 Fine-tuning with Example Aggregation

Simply combining the separately-trained networks together results in low performance. We perform additional fine-tuning with the two stages combined, and introduce a data augmentation method to learn to recover from error propagation.

Error Propagation Executing a sequence of instructions is susceptible to error propagation, where an agent fails to correctly complete an instruction, and because of it also fails on the following ones. While the collaborative, turn-switching setup allows the leader to adjust their plan following a follower mistake, leaders often strategically issue multiple instructions to use the available follower steps optimally. Given an agent failure, subsequent instructions may not align with the state of the world resulting from the follower's error. In supervised learning, we do not have the opportunity to learn to recover from such errors, even when it is relatively simple. This usually requires exploration. However, conventional frameworks like reinforcement learning (RL) or imitation learning (IL) are poorly suitable. In a live interaction, when an agent makes a mistake (e.g., selecting the wrong card), the leader is likely to adjust their actions. Because of this, in a recorded interaction, which contains the leader actions following a correct execution, it is not possible to reliably compute an RL reward for states following erroneous executions. For similar reasons, we cannot compute an IL oracle.

We identify two classes of erroneous states in CEREALBAR: (a) not selecting the correct set of cards; and (b) finishing with the right card selection, but stopping at the wrong position.[5] Case (a) requires to modify the model, for example to know when to skip instructions that refer to a state that is no longer possible. We leave this case for future work. We address case (b) by augmenting the data with new examples that are aggregated during learning. Our process is similar to DAGGER (Ross et al., 2011). We alternate between: (a) collecting new training examples using a heuristic oracle, and (b) performing model updates. We generate training examples that demonstrate recovery by starting in an incorrect initial position for an instruction, having arrived there by executing the previous instruction. We train our model to distinguish between the reasoning required for generating implicit actions to correct errors, and explicit actions directly mentioned in the instruction.

Learning with Example Aggregation We alternate between aggregating a new set of recovery examples $\mathcal{D}'$ and updating our parameters. At each epoch, we first use the current policy to create new training examples. We run inference for each example $\bar{I}^{(i,j)}$ in $\mathcal{D}$, the original training set, using the current policy.[6] We compare the state s' at the end of execution to the final state in $\bar{I}^{(i,j)}$ to generate an error-recovery example $\bar{I}'^{(i,j+1)}$ for the subsequent example $\bar{I}^{(i,j+1)}$. We only generate such examples if the position or rotation of the

[4] Additional details are in Appendix E.1.

[5] See Appendix E.2 for further discussion of the two cases.

[6] We do not perform inference for the last instruction in an interaction, as there is no subsequent example for which to generate a new example.

agent are different, and there are no other difference between the states. Starting from s', we generate the shortest-path sequence of actions that: (a) changes the cards as specified in $\bar{I}^{(i,j+1)}$, and (b) executes `DONE` in the same position as in $\bar{I}^{(i,j+1)}$. We then create $\bar{I}'^{(i,j+1)}$ using $\bar{I}^{(i,j+1)}$ and the new sequence of state-action pairs, and add it to $\mathcal{D}'$.[7]

Given the original set of examples $\mathcal{D}$ and the aggregate examples $\mathcal{D}'$ we update our model parameters. We randomly sample without replacement at most $\sum_{i=1}^{N} M^{(i)}$ examples, the size of $\mathcal{D}$, from $\mathcal{D}'$. We use all the examples in $\mathcal{D}$ and the sampled examples to do a single parameter update epoch. We limit the number of examples from $\mathcal{D}'$ to maintain the effect of the original data.

Optimizing with Implicit Action Prediction The examples we generate during aggregation often include sequences of state-action pairs that do not align with the instruction, for example when a mentioned spatial relation is incorrect from the new starting position. Such examples require reasoning differently about the text and world state than with the original data. We identify such examples in $\mathcal{D}'$ by comparing their follower starting position to the starting position in the original corresponding example in $\mathcal{D}$. If the distance is over two, we treat the examples as requiring implicit actions (Artzi and Zettlemoyer, 2013). All other examples, including all original examples in $\mathcal{D}$ are considered as not requiring implicit reasoning. We encourage the model to reason differently about these examples with a discriminator that classifies if the example requires implicit reasoning or not using the internal activations of LINGUNET.

The discriminator classifies each of the L layers in LINGUNET for implicit reasoning. The goal is to encourage implicit reasoning at all levels of reasoning in the first stage. The probability of implicit reasoning for each LINGUNET layer l is:

$$p(\text{IMPLICT} = 1 \mid l, s_1, \bar{x}) = \begin{cases} \sigma(\text{AVGPOOL}(\mathbf{G}_1 * \mathbf{K}_1^{\text{IMP}})) & l = 1 \\ \sigma(\text{AVGPOOL}(\mathbf{H}_l * \mathbf{K}_l^{\text{IMP}})) & l > 1 \end{cases} ,$$

where $\mathbf{K}_l^{\text{IMP}}$ are 1×1 learned kernels and AVGPOOL does average pooling. We define a cross-entropy loss $\mathcal{L}_{\text{IMP}}$ that averages across the L layers. The complete fine-tuning loss is:

$$\mathcal{L}(\theta_1, \theta_2) = \mathcal{L}_1(\theta_1) + \mathcal{L}_2(\theta_2) + \lambda_{\text{IMP}} \mathcal{L}_{\text{IMP}}(\theta_1) \ .$$

[7]Appendix E.2 describes this process.

6 Cascaded Evaluation

Sequential instruction scenarios are commonly evaluated using recorded interactions by executing individual instructions or executing complete interactions starting from their beginning (e.g., Chen and Mooney, 2011; Long et al., 2016). Both have limitations. Instruction-level metrics ignore error propagation, and do not accurately reflect the system's performance. In contrast, interaction-level metrics do consider error propagation and capture overall system performance well. However, they poorly utilize the test data, especially when performance is relatively low. When early failures lead to unexpected world states, later instructions become impossible to follow, and measuring performance on them is meaningless. For example, with our best-performing model, 82% of development instructions become impossible due cascading errors when executing complete interactions.

The two measures may also fail to distinguish models. For example, consider an interaction with three instructions. Two models, A and B, successfully execute the third instruction in isolation, but fail on the two others. They also both fail when executing the entire interaction starting from the beginning. According to common measures, the models are equal. However, if model B can actually recover from failing on the second instruction to successfully execute the third, it means it is better than model A. Both metrics fail to reflect this.

We propose *cascaded evaluation*, an evaluation protocol for sequential instruction using static corpora. Our method utilizes all instructions during testing, while still accounting for the effect of error propagation. Unlike instruction-level evaluation, cascaded evaluation executes the instructions in sequence. However, instead of starting of starting only from the start state of the first instruction, we create separate examples for starting from the starting state of each instruction in the interaction and continuing until the end of the interaction. For example, given a sequence of three instructions $\langle 1, 2, 3 \rangle$ we will create three examples: $\langle 1, 2, 3 \rangle$, $\langle 2, 3 \rangle$, and $\langle 3 \rangle$. To evaluate performance in CEREALBAR, we compute two statistics using cascaded evaluation: the proportion of the remaining instructions followed successfully, and the proportion of potential points scored. We only consider the remaining instructions and points left to achieve in the example. For example, for the sequence $\langle 2, 3 \rangle$, we will subtract any points achieved before the second instruction to compute the proportion of potential points scored. Appendix F describes cascaded evaluation formally.

7 Experimental Setup

Data We collect 1,202 human-human interactions using Mechanical Turk, split into train (960 games), development (120), and test (122). Appendix C details data collection and statistics.

Recorded Interactions Metrics We evaluate instruction-level, interaction-level, and cascaded (Section 6) performance. We allow the follower ten steps per turn, and interleave the actions taken by the leader during each turn in the recorded interaction. Instruction execution often crosses turns. At the instruction-level, we evaluate the mean *card state accuracy* comparing the state of the cards after inference with the correct card state, *environment state accuracy* comparing both cards and the agent's final position, and *action sequence accuracy* comparing the generated action sequence with the correct action sequence. For complete interactions, we measure mean *full game points*. Finally, for cascaded evaluation, we measure the mean proportion of instructions correctly executed and of possible points scored.

Human Evaluation We perform evaluation with human leaders, comparing our model and human followers. Workers are told they will work with a human or an automated follower, but are not told which in each game. We evaluate both human (105 games) and automated agents at the same time (109 games). We evaluate the game scores, and also elicit free-form feedback.

Systems We evaluate three systems: (a) the full model; (b) SEQ2SEQ+ATTN:[8] sequence-to-sequence with attention; and (c) a static oracle that executes the gold sequence of actions in the recorded interaction. We report mean and standard deviation across three trials for development results. We ablate model and learning components, and additionally evaluate the action generator with access to gold plans.[9] On the test set and for human evaluation, we use the model with the highest proportion of points scored. We provide implementation and learning details in Appendix G.

8 Results

Table 1 shows development and test results, including ablations. We consider the proportion of points scored computed with cascaded evaluation as the main metric. Our complete approach significantly outperforms SEQ2SEQ+ATTN. Key to this difference is the added structure within the model and the direct supervision on it. The results also show the large remaining gap to the static oracle.[10]

Our results show how considering error propagation for all available instructions in cascaded evaluation guides different design choices. For example, example aggregation and the implicit discriminator lower performance according to instruction-level metrics, which do not consider error propagation. We see a similar trend for the implicit discriminator when looking at full game points, an interaction-level metric that does not account for performance on over 80% of the data because of error propagation. In contrast, the proportion of points scored computed using cascaded evaluation shows the benefit of both mechanisms.

Our ablations demonstrate the benefit of each model component. All four distributions help. Without the trajectory distribution (– Trajectory distribution), performance drops almost to the level of SEQ2SEQ+ATTN. This indicates the action predictor is not robust enough to construct a path given only the three other disjoint distributions. While the predicted trajectory distribution contains all information necessary to reach the correct cards and goal location, the other three distributions further improve performance. This is likely because redundancy with the trajectory distribution makes the model more robust to noisy predictions in the trajectory distribution. For example, the GOAL distribution guides the agent to move towards goal cards even if the predicted trajectory is discontinuous. The action generation recurrence is also critical (– Action recurrence), allowing the agent to keep track of which locations it already passed when navigating complex paths that branch, loop, or overlap with themselves.

While we observe that each stage separately performs well after pretraining, combining them without fine-tuning (– Fine-tuning) leads to low performance because of the shift in the second stage input. Providing the gold distributions to the action generator illustrates this (+ Gold plan). Removing early goal auxiliary loss $\mathcal{L}_{G'}$ (Section 5.1) leads to a slight drop in performance on all metrics (– Early goal auxiliary). Learning with aggregated recovery examples helps the

[8] This baseline is similar to Mei et al. (2016).

[9] We do not measure interaction-level metrics with gold plans as they are only available for the gold start positions.

[10] Appendix F explains the static oracle performance.

System	Card State Acc.	Env. State Acc.	Action Seq. Accuracy	Full Game Points	Prop. Instr. Followed	Prop. Points Scored
Development Results & Ablation Analysis						
Full model	58.2±0.5	32.6±0.8	15.8±0.5	0.66±0.1	20.5±1.2	18.1±0.8
– Trajectory distribution	38.5±2.7	10.1±2.7	5.5±2.6	0.29±0.02	10.0±0.9	7.9±0.7
– GOAL distribution	56.2±1.5	30.8±0.4	14.9±0.3	0.66±0.09	17.9±1.0	15.9±1.3
– AVOID distribution	57.0±0.3	32.6±1.6	15.4±1.3	0.63±0.04	18.8±1.5	17.8±0.7
– NOPASS distribution	59.2±0.5	32.0±0.8	15.0±0.5	0.70±0.03	18.4±0.9	16.6±0.9
– Action recurrence	42.3±1.5	16.7±1.2	10.0±0.7	0.42±0.03	12.8±1.7	10.7±0.5
– Fine-tuning	43.6±1.9	8.5±1.1	4.5±0.5	0.65±0.09	14.1±1.3	9.2±0.9
– Early goal auxiliary	57.2±2.3	31.2±1.7	14.9±1.6	0.65±0.05	17.9±1.1	16.5±0.7
– Example aggregation	59.4±1.8	32.0±1.0	15.7±0.6	0.65±0.09	20.4±1.4	16.5±0.4
– Implicit discriminator	57.5±2.1	32.7±1.0	16.4±0.3	0.70±0.02	18.8±1.8	16.7±0.6
– Instructions	15.5±1.5	2.7±1.5	1.2±1.2	0.24±0.07	4.4±1.0	4.6±0.7
+ Gold plan	87.4±0.5	80.2±0.2	63.4±0.2	–	–	–
SEQ2SEQ+ATTN	35.3±0.8	11.1±0.5	9.4±0.5	0.20±0.04	8.8±0.1	6.3±0.1
Static oracle	99.7	99.7	100.0	6.58	98.5	97.9
Test Results						
Full model	58.4	32.1	15.6	0.62	15.4	17.9
SEQ2SEQ+ATTN	37.3	10.8	8.5	0.22	8.7	6.5
Static oracle	99.7	99.7	100.0	6.66	96.8	95.6

Table 1: Development and test results on all systems, including ablation results.

model to learn to recover from errors in previous instructions and increases the proportion of points scored (– Example aggregation). However, without the implicit reasoning discriminator (– Implicit discriminator), the additional examples make learning too difficult, and do not help. Finally, removing the language input (– Instructions) significantly decreases performance, showing that the data is relatively robust to observational biases and language is necessary for the task.

In the human evaluation, we observe a mean of 6.2 points (max of 14) with our follower model, compared to 12.7 (max of 20) with human followers. While this shows there is much room for improvement, it illustrates how human leaders adapt and use the agent effectively. One key strategy of adaptation is to use simplified language that fits the model better. This includes shorter instructions, with 8.5 tokens on average with automated followers compared to 12.3 with humans, and a smaller vocabulary, 578 word types with automated followers and 1037 with humans. In general, human leaders commented that they are able to easily distinguish between automated and human followers, and find working with the automated agent frustrating.

9 Discussion

Our human evaluation highlights several directions for future work. While human leaders adapt to the agent, scoring up to 14 points, there remains a significant gap to collaborations with human followers. Reported errors include getting stuck behind objects, selecting unmentioned cards, going in the wrong direction, and ignoring instructions. At least one worker developed a strategy that took advantage of the agent's full observability, writing instructions with only simple card references. An important direction for future work is to remove our full observability assumption. Other future directions include experimenting with using the interaction history, expanding the learning example aggregation to error cases beyond incorrect start positions, and making agent reasoning interpretable to reduce user frustration. CEREALBAR also provides opportunities to study pragmatic reasoning for language understanding (Andreas and Klein, 2016; Fried et al., 2018; Liang et al., 2019). While we currently focus on language understanding by limiting the communication to be unidirectional, bidirectional communication would allow for more natural and efficient collaborations (Potts, 2012; Ilinykh et al., 2019). CEREALBAR could be easily adapted to allow bidirectional communication, and provide a platform to study challenges in language generation.

Acknowledgments

This research was supported by the NSF under Grant No. 1750499, a Google Focused Award, an AI2 KSC Award, a Workday Faculty Award, Unity, and an Amazon Cloud Credits Grant. This material is based on work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-1650441. We thank Valts Blukis, Jin Sun, and Mark Yatskar for comments and suggestions, the workers who participated in our data collection, and the reviewers.

References

Anne H. Anderson, Miles Bader, Ellen Gurman Bard, Elizabeth Boyle, Gwyneth Doherty, Simon Garrod, Stephen Isard, Jacqueline Kowtko, Jan McAllister, Jim Miller, Catherine Sotillo, Henry S. Thompson, and Regina Weinert. 1991. The HCRC map task corpus. *Language and Speech*, 34.

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian D. Reid, Stephen Gould, and Anton van den Hengel. 2018. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3674–3683.

Jacob Andreas, Anca Dragan, and Dan Klein. 2017. Translating neuralese. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 232–242.

Jacob Andreas and Dan Klein. 2016. Reasoning about pragmatics with neural listeners and speakers. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1173–1182.

Jacob Arkin, Matthew R. Walter, Adrian Boteanu, Michael E. Napoli, Harel Biggie, Hadas Kress-Gazit, and Thomas M. Howard. 2017. Contextual awareness: Understanding monologic natural language instructions for autonomous robots. In *IEEE International Symposium on Robot and Human Interactive Communication*, pages 502–509.

Yoav Artzi, Dipanjan Das, and Slav Petrov. 2014. Learning compact lexicons for CCG semantic parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1273–1283.

Yoav Artzi and Luke Zettlemoyer. 2011. Bootstrapping semantic parsers from conversations. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 421–432.

Yoav Artzi and Luke Zettlemoyer. 2013. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association of Computational Linguistics*, 1:49–62.

Valts Blukis, Nataly Brukhim, Andrew Bennett, Ross A. Knepper, and Yoav Artzi. 2018a. Following high-level navigation instructions on a simulated quadcopter with imitation learning. In *Proceedings of the Robotics: Science and Systems Conference*.

Valts Blukis, Dipendra Misra, Ross A. Knepper, and Yoav Artzi. 2018b. Mapping navigation instructions to continuous control actions with position visitation prediction. In *Proceedings of the Conference on Robot Learning*.

Alexander Broad, Jacob Arkin, Nathan Ratliff, Thomas Howard, and Brenna Argall. 2017. Real-time natural language corrections for assistive robotic manipulators. *The International Journal of Robotics Research*, 36(5-7):684–698.

David L. Chen and Raymond J. Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the National Conference on Artificial Intelligence*.

Howard Chen, Alane Suhr, Dipendra Misra, Noah Snavely, and Yoav Artzi. 2019. Touchdown: Natural language navigation and spatial reasoning in visual street environments. In *IEEE Conference on Computer Vision and Pattern Recognition*.

Alex Djalali, David Clausen, Sven Lauer, Karl Schultz, and Christopher Potts. 2011. Modeling expert effects and common ground using questions under discussion. In *AAAI Fall Symposium: Building Representations of Common Ground with Intelligent Agents*.

Alex Djalali, Sven Lauer, and Christopher Potts. 2012. Corpus evidence for preference-driven interpretation. In *Logic, Language and Meaning*, pages 150–159.

Felix Duvallet, Thomas Kollar, and Anthony Stentz. 2013. Imitation learning for natural language direction following through unknown environments. In *IEEE International Conference on Robotics and Automation*, pages 1047–1053.

Jeffrey L. Elman. 1990. Finding structure in time. *Cognitive Science*, 14:179–211.

Katrina Evtimova, Andrew Drozdov, Douwe Kiela, and Kyunghyun Cho. 2017. Emergent communication in a multi-modal, multi-step referential game. In *Proceedings of the International Conference on Learning Representations*.

Daniel Fried, Jacob Andreas, and Dan Klein. 2018. Unified pragmatic models for generating and following instructions. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1951–1963.

He He, Anusha Balakrishnan, Mihail Eric, and Percy Liang. 2017. Learning symmetric collaborative dialogue agents with dynamic knowledge graph embeddings. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1766–1776.

He He, Derek Chen, Anusha Balakrishnan, and Percy Liang. 2018. Decoupling strategy and generation in negotiation dialogues. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 2333–2343.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9.

Hengyuan Hu, Denis Yarats, Qucheng Gong, Yuandong Tian, and Mike Lewis. 2019. Hierarchical decision making by generating and following natural language instructions. *CoRR*, abs/906.00744.

Nikolai Ilinykh, Sina Zarrieß, and David Schlangen. 2019. MeetUp! A corpus of joint activity dialogues in a visual environment. *CoRR*, abs/1907.05084.

Mohit Iyyer, Wen-tau Yih, and Ming-Wei Chang. 2017. Search-based neural structured learning for sequential question answering. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1821–1831.

Jin-Hwa Kim, Nikita Kitaev, Xinlei Chen, Marcus Rohrbach, Yuandong Tian, Dhruv Batra, and Devi Parikh. 2019. CoDraw: Collaborative Drawing as a Testbed for Grounded Goal-driven Communication. *CoRR*, abs/1704.04517.

Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations*.

Theodora Koulouri and Stanislao Lauria. 2009. Exploring miscommunication and collaborative behaviour in human-robot interaction. In *Proceedings of the SIGDIAL Conference*, pages 111–119.

Mike Lewis, Denis Yarats, Yann Dauphin, Devi Parikh, and Dhruv Batra. 2017. Deal or no deal? End-to-end learning of negotiation dialogues. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 2443–2453.

Claire Liang, Julia Proft, Erik Andersen, and Ross A. Knepper. 2019. Implicit communication of actionable information in human-AI teams. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, pages 95:1–95:13.

Karen E. Lochbaum. 1998. A collaborative planning model of intentional structure. *Computational Linguistics*, 24(4):525–572.

Reginald Long, Panupong Pasupat, and Percy Liang. 2016. Simpler context-dependent logical forms via model projections. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1456–1465.

Matthew MacMahon, Brian Stankiewics, and Benjamin Kuipers. 2006. Walk the talk: Connecting language, knowledge, action in route instructions. In *Proceedings of the National Conference on Artificial Intelligence*.

Hongyuan Mei, Mohit Bansal, and Matthew R. Walter. 2016. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Scott Miller, David Stallard, Robert Bobrow, and Richard Schwartz. 1996. A fully statistical approach to natural language interfaces. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 55–61.

Dipendra Misra, Andrew Bennett, Valts Blukis, Eyvind Niklasson, Max Shatkhin, and Yoav Artzi. 2018. Mapping instructions to actions in 3D environments with visual goal prediction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 2667–2678.

Dipendra Misra, John Langford, and Yoav Artzi. 2017. Mapping instructions and visual observations to actions with reinforcement learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1004–1015.

Christopher Potts. 2012. Goal-driven answers in the Cards dialogue corpus. In *Proceedings of the West Coast Conference on Formal Linguistics*, pages 1–20.

Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. 2011. A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*.

Candace L. Sidner, Carolyn Boettner, and Charles Rich. 2000. Lessons learned in building spoken language collaborative interface agents. In *ANLP-NAACL Workshop: Conversational Systems*.

Alane Suhr and Yoav Artzi. 2018. Situated mapping of sequential instructions to actions with single-step reward observation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 2072–2082.

Alane Suhr, Srinivasan Iyer, and Yoav Artzi. 2018. Learning to map context-dependent sentences to executable formal queries. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2238–2249.

Stephanie Tellex, Thomas Kollar, Steven Dickerson, Matthew R. Walter, Ashis Gopal Banerjee, Seth Teller, and Nicholas Roy. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the National Conference on Artificial Intelligence*.

Jesse Thomason, Shiqi Zhang, Raymond Mooney, and Peter Stone. 2015. Learning to interpret natural language commands through human-robot dialog. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Takuma Udagawa and Akiko Aizawa. 2019. A natural language corpus of common grounding under continuous and partially-observable context. In *Proceedings of the Conference on Artificial Intelligence*.

Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. 2017. Improved texture networks: Maximizing quality and diversity in feed-forward stylization and texture synthesis. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 4105–4113.

Andreas Vlachos and Stephen Clark. 2014. A new corpus and imitation learning framework for context-dependent semantic parsing. *Transactions of the Association for Computational Linguistics*, 2:547–560.

Harm de Vries, Kurt Shuster, Dhruv Batra, Devi Parikh, Jason Weston, and Douwe Kiela. 2018. Talk the Walk: Navigating New York City through grounded dialogue. *arXiv preprint arXiv:1807.03367*.

Sida I. Wang, Percy Liang, and Christopher D. Manning. 2016. Learning language games through interaction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 2368–2378.

Jason Williams, Antoine Raux, Deepak Ramachandran, and Alan Black. 2013. The dialog state tracking challenge. In *Proceedings of the SIGDIAL Conference*, pages 404–413.

Ronald J. Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1:270–280.

A CerealBar Game Design

This appendix supplements Section 2 with further game design details and discussion of the reasoning behind them.

World View Figure 3 shows the leader's point of view, and Figure 4 shows the follower's. The leader observes the entire environment, while the follower only has access to a restricted first person view. The leader can also toggle to an overhead view to see obstructed cards using the camera button, and has access to the follower's current view to aide them in writing instructions that make sense to the follower. Selected cards are outlined in blue for both players. Invalid selections appear in red for the leader only. This setup makes the follower dependent on the leader, limits the follower ability to plan the card collection strategy, and encourages collaboration.

Game Progression The two players switch control of the game by taking turns. During each turn, the follower can take ten ($\Psi_f = 10$) steps while the leader can take five ($\Psi_l = 5$). Allowing the follower more steps than the leader incentivizes delegating lengthier tasks to the follower, such as grabbing multiple cards per turn or moving further away. We do not count actions which do not change the player's location or rotation, such as moving forward into an obstacle, against this limit. We additionally limit the amount of time each player has per turn. This requires players to move quickly without frustrating their partner by taking a long time, and additionally limits the maximum time per game. Both players begin with six turns each. The game ends when the players run out of turns.

The leader turn ends once they press the end turn button or after 45 seconds. The end turn button is disabled as long as there are no instructions in the follower queue to nudge the leader to use the follower if time allows it. The allotted 45 seconds allow the leader sufficient time to move, plan, and write instructions. During the leader's turn, they can add any number of new instructions to the queue.

The follower only receives control if there are instructions in the queue. If the queue is empty when the leader finishes their turn, the follower's turn is skipped, but the number of turns remaining still decreases. The follower's turn ends automatically when they run out of steps, after 15 seconds, or when they complete all instructions in the queue. During the follower's turn, they can mark any number of instructions as complete using the `DONE` action. The follower sees the current and previous instructions, even if there are more instructions in the queue. They must mark the current instruction as complete before seeing the next. This is done to simplify the reasoning available to the follower. For example, to avoid cases where the follower skips a command based on future ones. Because there may be more future instructions in the queue, this incentivizes the follower to not waste moves in the current instruction and be as efficient as possible. During data collection, this provides alignment of actions to instructions because it prohibits a follower from taking actions aligning with a future instruction without marking the current instruction as complete. Without instruction completion annotation, the problem of alignment between instructions and actions becomes much more difficult when processing the recorded interactions.

Scoring Points When a valid set is made, the selected cards disappear, and three cards are randomly generated and placed on the grid such that the new grid contains a least one valid set. The two players earn a point, and are given extra turns. The number of added turns decays as they complete more sets, eventually reaching zero added turns. The maximum possible number of turns in a game is 65. In the training data, 454 games reached this number of turns. Adding extra turns when a set is made allows us to collect more data from games that are going well. It also allows us to pay players based on the number of sets completed, and incentivizes them to play as well as possible. If a game is going poorly, e.g., if the pair fails to earn a point in the first six turns, the game will end early. However, if the game is going well, implying the pair is collaborating well, the game will continue for longer, and will contain a longer sequence of instructions.

B CerealBar Transition Function

The transition function in CerealBar $\mathcal{T} : \mathcal{S} \times \Gamma \times \mathcal{A} \to \mathcal{S} \times \Gamma$ is formally defined in Table 2. Each of the rules in the table is additionally associated with a domain over which it is not defined, for example when $\alpha =$ Follower and $a \in \mathcal{X}$ (i.e., the follower can not give instructions). The rules are:

Rule 1: When an instruction is issued, it is added to the end of the queue. This action does not

Rule No.	Domain	Definition		
1	$\forall \bar{x} \in \mathcal{X}, s \in \mathcal{S}$	$\mathcal{T}(s, \langle Q, \text{Leader}, \psi \rangle, \bar{x})$	$=$	$(s, \langle Q\bar{x}, \text{Leader}, \psi \rangle)$
2	$\forall s \in \mathcal{S}, \lvert Q \rvert \geq 1$	$\mathcal{T}(s, \langle Q, \text{Leader}, \psi \rangle, \texttt{DONE})$	$=$	$(s, \langle Q, \text{Follower}, \Psi_f \rangle)$
3	$\forall s \in \mathcal{S}, \lvert Q \rvert = 0$	$\mathcal{T}(s, \langle \langle\, \rangle, \text{Leader}, \psi \rangle, \texttt{DONE})$	$=$	$(s\langle Q, \text{Leader}, \Psi_l \rangle)$
4	$\forall a \in \mathcal{A}_w, s \in \mathcal{S}$	$\mathcal{T}(s, \langle Q, \text{Leader}, 1 \rangle, a)$	$=$	$(\mathcal{T}_w(s, a), \langle Q, \text{Leader}, 0 \rangle)$
5	$\forall s \in \mathcal{S}, \lvert Q \rvert > 1$	$\mathcal{T}(s, \langle \bar{x}Q, \text{Follower}, \psi \rangle, \texttt{DONE})$	$=$	$(s, \langle Q, \text{Follower}, \psi \rangle)$
6	$\forall s \in \mathcal{S}, \lvert Q \rvert = 1$	$\mathcal{T}(s, \langle Q, \text{Follower}, \psi \rangle, \texttt{DONE})$	$=$	$(s, \langle \langle\, \rangle, \text{Leader}, \Psi_l \rangle)$
7	$\forall a \in \mathcal{A}_w, s \in \mathcal{S}$	$\mathcal{T}(s, \langle Q, \text{Follower}, 1 \rangle, a)$	$=$	$(\mathcal{T}_w(s, a), \langle Q, \text{Leader}, \Psi_l \rangle)$
8	$\forall a \in \mathcal{A}_w, s \in \mathcal{S}$ $\forall \psi \in \mathbb{N}_{>1}$ $\forall \alpha \in \{\text{Leader}, \text{Follower}\}$	$\mathcal{T}\left(s, \langle \bar{Q}, \alpha, \psi \rangle, a\right)$	$=$	$\left(\mathcal{T}_w(s, a), \langle \bar{Q}, \alpha, \psi - 1 \rangle\right)$

Table 2: Definition of transition function $\mathcal{T}$. $\mathcal{T}_w$ is the world state transition function.

use a step, so the number of steps remaining ψ does not decrease. This rule is not defined when α = Follower because the follower cannot give an instruction.

Rule 2: When the leader ends their turn, and the queue is not empty, control switches to the follower, and the number of steps remaining in the turn is the maximum number for the follower Ψ_f.

Rule 3: When the leader ends their turn, and the queue is empty, control does not switch to the follower; instead, a new leader turn begins with Ψ_l available steps.

Rule 4: When the leader runs out of remaining steps, control does not immediately switch to the follower. This allows the leader to issue more instructions before manually ending their turn or when their time runs out.

Rule 5: When the follower marks an instruction as finished, and more instructions remain in the queue, the current instruction at the head of the queue is removed. This action does not use a step.

Rule 6: When the follower marks an instruction as finished, if the finished instruction was the last in the queue, control automatically switches to the leader with Ψ_l remaining steps.

Rule 7: When the follower runs out of steps in their turn, control immediately switches to the leader with Ψ_l remaining steps.

Rule 8: Both agents can take actions which modify the world state s. Each such action $a \in \mathcal{A}_w$ costs a step. We assume access to a domain-specific transition function, $\mathcal{T}_w : \mathcal{S} \times \mathcal{A}_w \rightarrow \mathcal{S}$, that describes how an environment action modifies the environment. There may exist combinations of states and actions for which $\mathcal{T}_w$ is not defined; for example, an agent moving forward onto an obstacle. Additionally, $\forall s \in \mathcal{S}$ and $a \in \mathcal{A}_w$, $\mathcal{T}(s, \langle Q, \text{Leader}, 0 \rangle, a)$ results in an invalid state because, while the leader can still issue instructions after running out of steps, they cannot move.

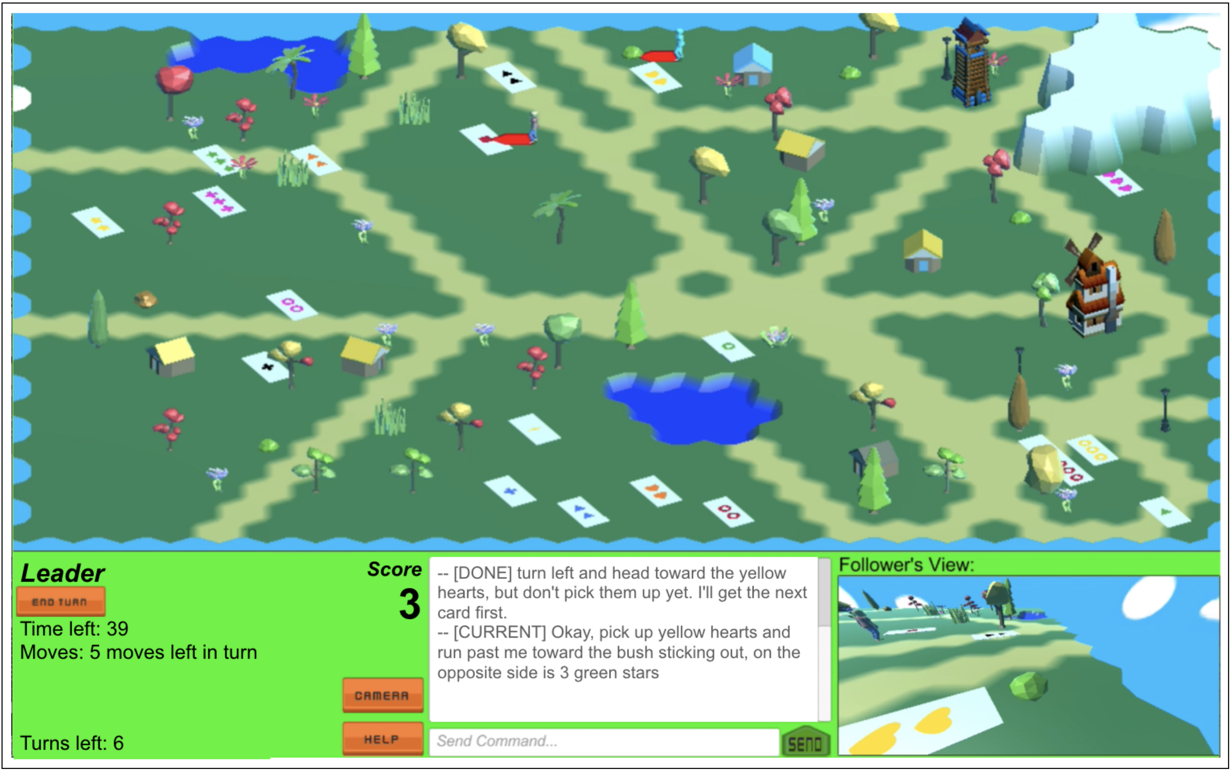
C Data Collection Details

Figures 3 and 4 show the leader's and follower's interfaces.

Crowdsourcing Management We use a qualification task to both teach workers how to play the game and to mark workers as qualified for our main task. We restrict those who can qualify to workers located in majority English-speaking countries with at least 90% approved HITs and at least 100 completed HITs. The qualification task has three components: an interactive tutorial for the leader role, an interactive tutorial for the follower role, and a short quiz about the gameplay. In both tutorials, turn-switching is disabled and workers have an unlimited number of moves to use to complete the tutorial. Each tutorial uses the same map. This allows us to pre-program instructions for the tutorials.

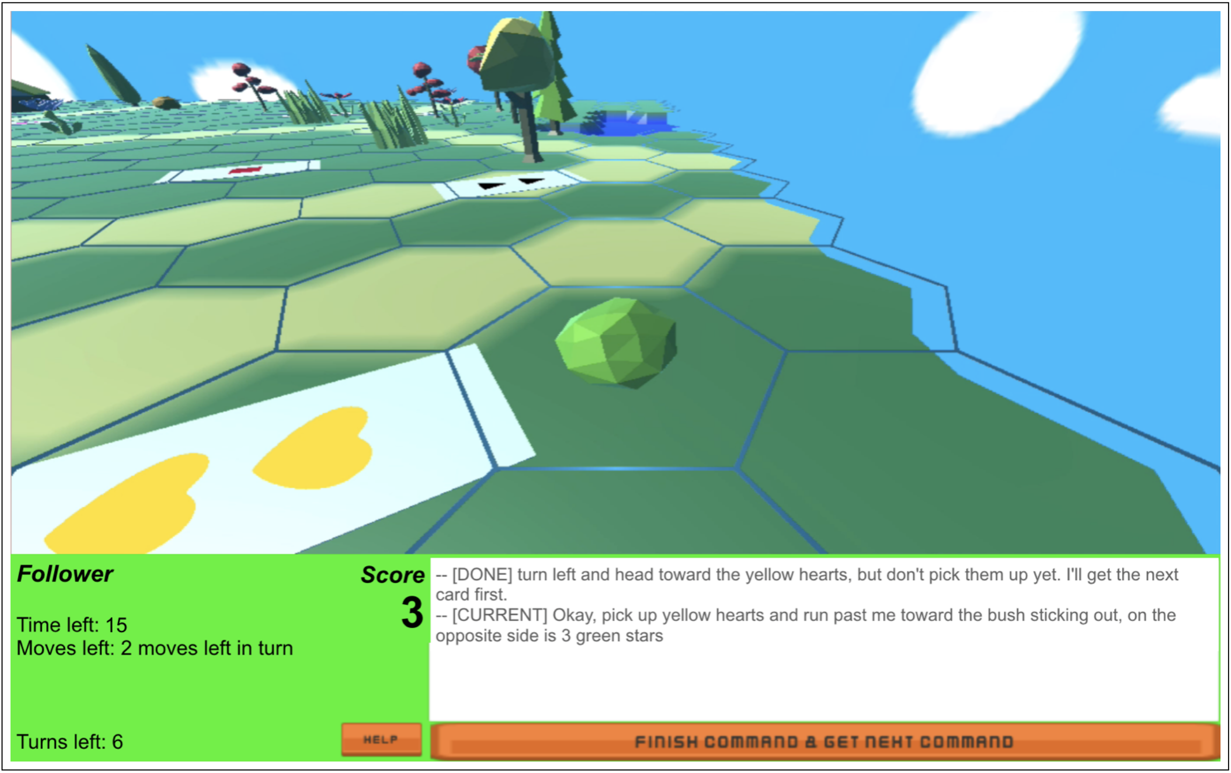
In the leader tutorial, the worker has access to the full game board. They are asked to send a command to the follower, and are instructed via in-game prompts to collect a specific set of cards. Finally, they are asked to collect two more sets in the environment that are valid. Workers who send a command and collect a total of three sets successfully complete this tutorial.

In the follower tutorial, the worker has access only to the follower view. Pre-written commands are issued to the worker, and they must follow them one-by-one to complete a set. The commands include an example of the leader correcting a set-planning mistake. If the worker marks



-- [DONE] turn left and head toward the yellow hearts, but don't pick them up yet. I'll get the next card first.
-- [CURRENT] Okay, pick up yellow hearts and run past me toward the bush sticking out, on the opposite side is 3 green stars

Figure 3: The CEREALBAR leader gameplay interface.


Follower
Time left: 15
Moves left: 2 moves left in turn
Turns left: 6
Score
3
HELP
FINISH COMMAND & GET NEXT COMMAND

-- [DONE] turn left and head toward the yellow hearts, but don't pick them up yet. I'll get the next card first.
-- [CURRENT] Okay, pick up yellow hearts and run past me toward the bush sticking out, on the opposite side is 3 green stars

Figure 4: The CEREALBAR follower gameplay interface.

all commands as finished and successfully collects one set, the follower tutorial is complete.

Finally, workers are asked to read the game instructions and complete a short quiz. They are asked questions regarding the validity of card sets, the responsibilities of both players, and how each game ends.

We maintain two groups of workers split by experience with the game, and use separate pools of HITs for each. A worker can join the expert pool if they have shown they understand how to play as a leader and as a follower through at least one game each. This allows new players to learn the game rules without frustrating expert players. At the end of data collection, 95 workers were in the expert pool while 169 were in the non-expert pool, for a total of 264 participating workers.

We pay workers a bonus per point they earn, increasing the bonus as more points are earned, in addition to a base pay of per game. We do not pay leaders and followers differently. The median game cost was $5.80.

The CEREALBAR Dataset In total, we collect 1,526 games played by both experts and non-experts. Of these, we keep 1,202 (78.8%) games, comprising 23,979 total instructions, discarding those where no instructions were complete, or where alignment between instructions and actions was suspected low-quality. For example, we removed interactions with a low proportion of instructions being marked as complete, or very long action sequences from the follower, both which indicate the follower did not properly complete instructions.

When splitting the data, we ensured the mean score between the three splits was roughly the same. Table 3 shows basic statistics of the data we collected after pruning. 82.6% of post-pruning games are from the expert pool. In the training set, the mean number of completed instructions is 19.9 and the median is 24.0. 83.3% of games have a score greater than zero. We include games with a score of zero if the alignment between instructions and actions is high-quality according to our pruning heuristics. The vocabulary size is computed by lowercasing all word types and tokenizing using the NLTK word tokenizer. Our dataset contains longer interactions than several existing datasets for sequential instruction following and interaction (e.g., Chen and Mooney, 2011; Long et al., 2016; He et al., 2017; de Vries et al., 2018; Kim et al., 2019; Hu et al., 2019; Udagawa and Aizawa, 2019), though still shorter than the Cards corpus (Djalali et al., 2011, 2012; Potts, 2012). Individual sentences are also longer than several similar corpora (e.g., Chen and Mooney, 2011; Djalali et al., 2011; Long et al., 2016; He et al., 2017; Hu et al., 2019).

	Mean	**Median**	**Max**
Total Game Time (m:s)	16:28	18:40	31:31
Score / Interaction	7.9	9.0	19
# Instr. / Interaction	19.9	24.0	40
# Tokens / Instr.	14.0	13.0	55
# Follower Actions / Instr.	8.5	8.0	50
# Interactions	1,202		
Vocabulary Size	3,641		

Table 3: Human-human games data statistics. All statistics except the number of examples are computed on the training set only.

D Model Architecture Details

LINGUNET Formal Description We provide a formal description of LINGUNET for reference only. LINGUNET was originally introduced by Misra et al. (2018) and Blukis et al. (2018a).

The input to LINGUNET are the environment representation $\mathbf{F}_0$ and instruction representation $\bar{\mathbf{x}}$. LINGUNET consists of three major stages: a series of convolutions on $\mathbf{F}_0$, a series of text-based convolutions derived from $\bar{\mathbf{x}}$, and a series of transposed convolutions to form a final prediction. The output of the LINGUNET is a feature map with the same width and height as $\mathbf{F}_0$. Each stage has the same number of operations, which we refer to as the depth L.

First, a series of L convolutional layers is applied to $\mathbf{F}_0$. Each layer at depth l is a sequence of two convolution operations separated by a leaky ReLU non-linearity:

$$\mathbf{F}_l = \textsc{Norm}(\textsc{ReLU}(\textsc{ReLU}(\mathbf{F}_{l-1} * \mathbf{K}_l^C) * \mathbf{K}_l^{C'})) \ .$$

We use a stride of two when convolving with $\mathbf{K}_l^{C'}$, and do not apply NORM when $l = L$.

In the second stage, the instruction representation $\bar{\mathbf{x}}$ is split into L segments $\bar{\mathbf{x}}_l$ such that $\bar{\mathbf{x}} = [\bar{\mathbf{x}}_1; \ldots; \bar{\mathbf{x}}_L]$ and segments have equal length. Each segment is mapped to a 1×1 kernel $\mathbf{K}_l^I$ using learned weights $\mathbf{W}_l^I$ and biases $\mathbf{b}_l^I$. $\mathbf{K}_l^I$ is normalized and used to convolve over $\mathbf{F}_l$:

$$\mathbf{G}_l = \textsc{Norm}(\mathbf{F}_l * ||\mathbf{K}_l^I||_2) \ .$$

As before, we do not apply NORM when $l = L$.

In the last stage, a series of transposed convolutions[11] are applied starting from the bottom layer

[11]We use $*^\top$ to represent the transposed convolution oper-

and gradually synthesizing a larger feature map. For $l > 1$:

$$\mathbf{H}_l = \textsc{Norm}(\textsc{ReLU}(\textsc{ReLU}([\mathbf{H}_{l+1};\mathbf{G}_l]*^\top\mathbf{K}_l^T)*^\top\mathbf{K}_l^{T\prime}))\ ,$$

where $[\mathbf{H};\mathbf{G}]$ indicates channel-wise concatenation of feature maps $\mathbf{H}$ and $\mathbf{G}$, $\mathbf{H}_{H+1}$ is a zero matrix, and $\textsc{Norm}$ is not applied when $l = L$. We use a stride of two when convolving with $\mathbf{K}_l^{T\prime}$. At the topmost layer of LINGUNET, a final transposed convolution is applied to form a feature map $\mathbf{H}_1'$:

$$\mathbf{H}_1' = [\mathbf{H}_2;\mathbf{G}_1] *^\top \mathbf{K}_1^T\ .$$

The top layer $\mathbf{H}_1' \in \mathbb{R}^{4\times W\times H}$ is split into the four planning distributions as the output of the LINGUNET.

Frames of Reference The world state is first embedded using a feature lookup and a text-conditioned kernel (Section 4; Input Representation). This feature map is rotated and centered to create $\mathbf{F}_0$, so that the agent's location when beginning to follow the instruction is in the center, and the agent is facing in a consistent direction. Therefore, LINGUNET (Section 4; Stage 1: Plan Prediction) operates over a feature map relative to the agent's frame of reference at the time of starting to follow the instruction.

The action generator (Section 4; Stage 2: Action Generation) also operates on feature maps relative to the agent's frame of reference, updated as the agent moves and turns in the environment changing its location and orientation. At each action generation prediction step, the concatenated planning distributions $\mathbf{P}$ are rotated, centered, and cropped around the agent's current orientation. This orientation is determined by the orientation when starting the instructions and the actions it has executed so far for the current instruction.

E Learning Details

E.1 Stage 1 Loss Computation

This section provides formal details of the loss computation used in Section 5.1. For ease of notation, we consider a single example $\bar{I} = \langle(s_1,\gamma_1,a_1),\ldots,(s_n,\gamma_n,a_n)\rangle$, where the instruction at the head of the queue $\bar{Q}$ is $\bar{x}$.

The loss of the visitation distribution $p(\rho \mid s_1,\bar{x})$ is:

$$\mathcal{L}_V(\theta_1) = -\sum_\rho p_V^*(\rho)\log p(\rho \mid s_1,\bar{x})\ ,$$

ation.

where the summation is over all positions ρ in the environment and $p_V^*(\rho)$ is proportional to the number of states $s_t \in \bar{I}$ where the follower is in position ρ.

We compute the goal and avoidance distribution losses only for positions that have cards:

$$\mathcal{L}_G(\theta_1) = -\tfrac{1}{W\times H}\sum_{\rho\in C} p_G^*(\rho)\log p(\mathrm{GOAL}=1 \mid \rho,s_1,\bar{x})$$
$$\mathcal{L}_A(\theta_1) = -\tfrac{1}{W\times H}\sum_{\rho\in C} p_A^*(\rho)\log p(\mathrm{AVOID}=1 \mid \rho,s_1,\bar{x})\ ,$$

where C is the set positions that contain cards, W is the width of the environment, and H is the height. We set $p_G^*(\rho)$ to 1 for all ρ that contain a card that the follower changed its selection status in $\bar{I}$, and 0 for all other positions. Similarly, we set $p_A^*(\rho)$ to 1 for all ρ that have cards that the follower does not change during the interaction $\bar{I}$, but zero for the initial position regardless of whether it contains a card.

The loss for the no passing distribution is:

$$\mathcal{L}_P(\theta_1) = -\tfrac{1}{W\times H}\sum_\rho p_P^*(\rho)\log p(\mathrm{NOPASS}=1 \mid \rho,s_1,\bar{x})\ ,$$

where $p_P^*(\rho)$ is 1 for all positions the agent cannot move onto, and zero otherwise.

The auxiliary goal-prediction loss is:

$$\mathcal{L}_{G'}(\theta_1) = -\tfrac{1}{W\times H}\sum_{\rho\in C} p_G^*(\rho)\log p_G'(\mathrm{GOAL}=1 \mid \rho,s_1,\bar{x})\ .$$

We compute the goal probability with the learned parameters $\mathbf{W}^{G'}$ and $\mathbf{b}^{G'}$:

$$p_G'(\mathrm{GOAL}=1 \mid \rho,s_1,\bar{x}) = \sigma(\mathbf{W}^{G'}\mathbf{S}_\rho' + \mathbf{b}^{G'})\ ,$$

where $\mathbf{S}_\rho'$ is the vector along the channel dimension for position ρ in the environment embedding tensor $\mathbf{S}'$.

E.2 Example Aggregation

Error Classes We identify two classes of erroneous states in CEREALBAR: (a) not selecting the correct set of cards specified by the instruction; and (b) finishing with the right card selection, but stopping at the wrong position. To recover from case (a), the agent could unselect cards it shouldn't have selected, or select cards it missed. Alternatively, the agent could recognize it has made an error, and instead stop and wait for the next leader instruction, anticipating a correction. However, learning this requires access to previous world states and instructions. We focus on modification of the learning algorithm using example aggregation, and leave this case for future work. We instead target class (b), and add a discriminator to the model to allow the model to learn different reasoning for examples that require implicit actions, as discussed in Section 5.2.

Creating Recovery Examples The oracle generates a sequence of state-action pairs to go from s', the incorrect initial state from the previous instruction, to state s_t at index t in the correct sequence such that s_t is either the first state in the sequence where a card's state changes, or if no cards are changed, the final state s_n. The oracle finds a sequence of state-action pairs expressing the shortest path s' to s_t. Finally, it appends the remainder of the correct state-action sequence starting from index t, $\langle (s_t, \gamma_t, a_t), \ldots, (s_n, \gamma_n, a_n) \rangle$.

If the correct sequence for $\bar{I}^{(i,j+1)}$ is $\langle s_n, \gamma_n, \texttt{DONE} \rangle$ (i.e., no action was done in the original example), we do not generate a new path, but instead use the state-action sequence $\langle s', \gamma', \texttt{DONE} \rangle$ as annotation for $\bar{I}'^{(i,j+1)}$. These examples are annotated as not requiring implicit reasoning.

During inference on the previous example $\bar{I}^{(i,j)}$, it is possible that some leader actions associated with that example may not be executed (i.e., if the follower predicted DONE too soon). If this happens, the leader must execute actions to 'catch up' to the follower in the generated recovery example. We first find the sequence of leader actions starting from the first leader turn associated with $\bar{I}^{(i,j)}$ that was not executed during inference, to the final leader turn associated with $\bar{I}^{(i,j+1)}$. When generating the recovery sequence $\bar{I}'^{(i,j+1)}$, we take into consideration this sequence as affecting the world states s. For example, suppose that the agent stops a turn early during inference, and the final leader's turn consisting of actions $\langle \texttt{FORWARD}, \texttt{FORWARD}, \texttt{FORWARD}, \texttt{DONE} \rangle$ was not executed. Instead of stopping in, for example, position $(3, 0)$, this may mean the leader has stopped in position $(0, 0)$. When creating the recovery example, the first world state s_0 shows the leader at position $(0, 0)$ rather than $(3, 0)$. To correct this, the recovery example will start with a leader turn, where the leader executes the sequence $\langle \texttt{FORWARD}, \texttt{FORWARD}, \texttt{FORWARD}, \texttt{DONE} \rangle$.

F Evaluation Details

Cascaded Evaluation To compute metrics using cascaded evaluation, we construct a set of cascaded evaluation examples from the original test set. We assume access to a test set of M recorded interactions $\{\bar{I}^{(i)}\}_{i=1}^{M}$, where each $\bar{I}^{(i)} = \left\langle \left(s_1^{(i)}, \gamma_1^{(i)}, a_1^{(i)}\right), \ldots, \left(s_{|\bar{I}|}^{(i)}, \gamma_{|\bar{I}|}^{(i)}, a_{|\bar{I}|}^{(i)}\right) \right\rangle$. For each instruction $\bar{x}_j$ in $\bar{I}^{(i)}$, we create an example $\bar{I}_C^{(i,j)} = \left\langle \left(s_{j'}^{(i)}, \gamma_{j'}^{(i)}, a_{j'}^{(i)}\right), \ldots, \left(s_{|\bar{I}|}^{(i)}, \gamma_{|\bar{I}|}^{(i)}, a_{|\bar{I}|}^{(i)}\right) \right\rangle$, where j' is the first follower step of executing $\bar{x}_j$. We treat each $\bar{I}_C^{(i,j)}$ as a separate example. For each metric, we report the proportion of the maximum value possible for each $\bar{I}_C^{(i,j)}$, and average across all examples $\bar{I}_C^{(i,j)}$. When computing the proportion of instructions followed in cascaded evaluation, the maximum possible for example $\bar{I}_C^{(i,j)}$ is the number of remaining instructions $N - j$ where N is the number of instructions in $\bar{I}^{(i)}$. When computing the proportion of points scored, we subtract the points scored in the game before step j to only account for points possible in the instructions present in $\bar{I}_C^{(i,j)}$.

Performance of the Static Oracle The static oracle does not have perfect performance. This is because the follower's turn ended before all ten steps were used in some recorded interactions. During evaluation, however, we allow the follower to move for all ten available steps. This sometimes leads to misalignment between leader and follower actions. This means some expected sets can not be completed.

G Implementation and Hyperparameters

Hyperparameters We tune hyperparameters on the development set. We use a word embedding size of 64, and encode instructions into a vector of length 64 using a single-layer RNN with LSTM units. We lowercase words in the vocabulary and map all words with a frequency of one in the training set to a single out-of-vocabulary token. We use a hex property embedding size of 32. $\mathbf{S}'$ has four channels. The text-based kernels map to a feature map with 24 channels. The convolution and transpose convolution phases of LingUNet use kernel sizes of three.

The action generator uses a forward RNN with a single layer consisting of 128 LSTM hidden units. The action embedding size is 32. We rotate, transform, and crop the input plan distribution to a $4 \times 5 \times 5$ feature map around the agent's current position and rotation for each generated action. CNN^P maps the cropped distributions to a feature map with eight channels, and has a kernel size of three and stride of one. During fine-tuning, each $\mathbf{K}_l^{\text{IMP}}$ does not have biases. For all LSTMs, we initialize the hidden state $\mathbf{h}_0$ as a zero vector. For brevity, cell memory $\mathbf{c}^D$, also initialized as a zero vector, is omitted from RNN descriptions.

Learning The plan prediction stage (Stage 1) includes the following parameters and parameterized components: $\phi^{\mathcal{X}}$, RNN$^{\mathcal{X}}$, $\phi^{\mathcal{S}}$, $\mathbf{W}_s$, $\mathbf{b}_s$, and LINGUNET. The action generation stage (Stage 2) includes the following parameters and parameterized components: CNN^P, $\mathbf{W}_1^P$, $\mathbf{W}_2^P$, $\mathbf{b}_1^P$, $\mathbf{b}_2^P$, $\phi^{\mathcal{A}}$, RNN$^{\mathcal{A}}$, $\mathbf{W}^{\mathcal{A}}$, and $\mathbf{b}^{\mathcal{A}}$. We add the following parameters for the early goal prediction auxiliary objective and implicit reasoning discriminator $\mathbf{W}^{G'}$, $\mathbf{b}^{G'}$, and $\mathbf{K}_l^{\text{IMP}}$, $1 < l < L$.

For pretraining Stage 1, we use a learning rate of 0.0075 using ADAM (Kingma and Ba, 2014) and an L2 coefficient of 10^-6. For pretraining Stage 2 and during fine-tuning, we use a learning rate of 0.001 and ADAM with no L2 regularization. For pretraining Stage 1 and during fine-tuning, $\lambda_V = 1$, $\lambda_G = 1$, $\lambda_A = 0.1$, $\lambda_P = 0.1$, and $\lambda_{G'} = 1$. During fine-tuning, $\lambda_{\text{IMP}} = 0.7$. During evaluation, we limit the maximum action sequence length to 25.

For all experiments, we keep 5% of the training data as held-out from parameter updates and used as a validation set. We use patience for stopping during pretraining of the plan predictor and the action generator (Section 5.1). We start with a patience of 10, which increases by a factor of 1.01 each time the stopping metric improves on the validation set. For plan prediction training, we use patience on the validation set accuracy of predicted goal locations. We compute goal location predictions by finding all positions ρ such that $p(\text{GOAL} = 1 \mid \rho, s_1, \bar{x}) \geq 0.5$. For action generation, we stop when card-state accuracy reaches a maximum on the validation set. For fine-tuning (Section 5.2), we stop training after 25 epochs, and choose the epoch that maximizes the proportion of points scored computed using cascaded evaluation (Section 6) on the validation set.

SEQ2SEQ+ATTN Baseline We embed the sentence tokens into 64-dimensional vectors, and compute a sentence representation using a single-layer RNN with 64 LSTM hidden units. We embed each position in the environment with a learned embedding function $\phi^{\mathcal{S}}$ mapping to a vector of size 32. The resulting feature map is put through four convolutional layers separated by leaky ReLU non-linearities. Each convolutional layer has a stride of two and divides the number of channels in half. The output of the last convolutional layer is flattened to a vector.

We initialize the decoder hidden state to a zero-vector. In each timestep we pass in the concatenation of the embedding of the previous output, the embedded environment vector, and the previous result of the attention computation on the sentence. We take the initial attention result to be a zero vector. We compute the attention over the sentence hidden states using the dot product of hidden state with the current hidden state in the decoder RNN. The resulting attention state is concatenated with the decoder hidden state and the embedded environment vector, put through a leaky ReLU non-linearity, and and finally through a single fully-connected layer to predict probabilities over actions.

We train the model using teacher forcing and apply the same learning rate, optimizer, and stopping criteria as the fine-tuning experiments.