Appendix: 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

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

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

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 $\mathcal{T} : \mathcal{S} \times \Gamma \times \mathcal{A} \rightarrow \mathcal{S} \times \Gamma$ 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 use a step, so the number of steps remaining ψ does not decrease. This rule is not defined when $\alpha = \text{Follower}$ 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.

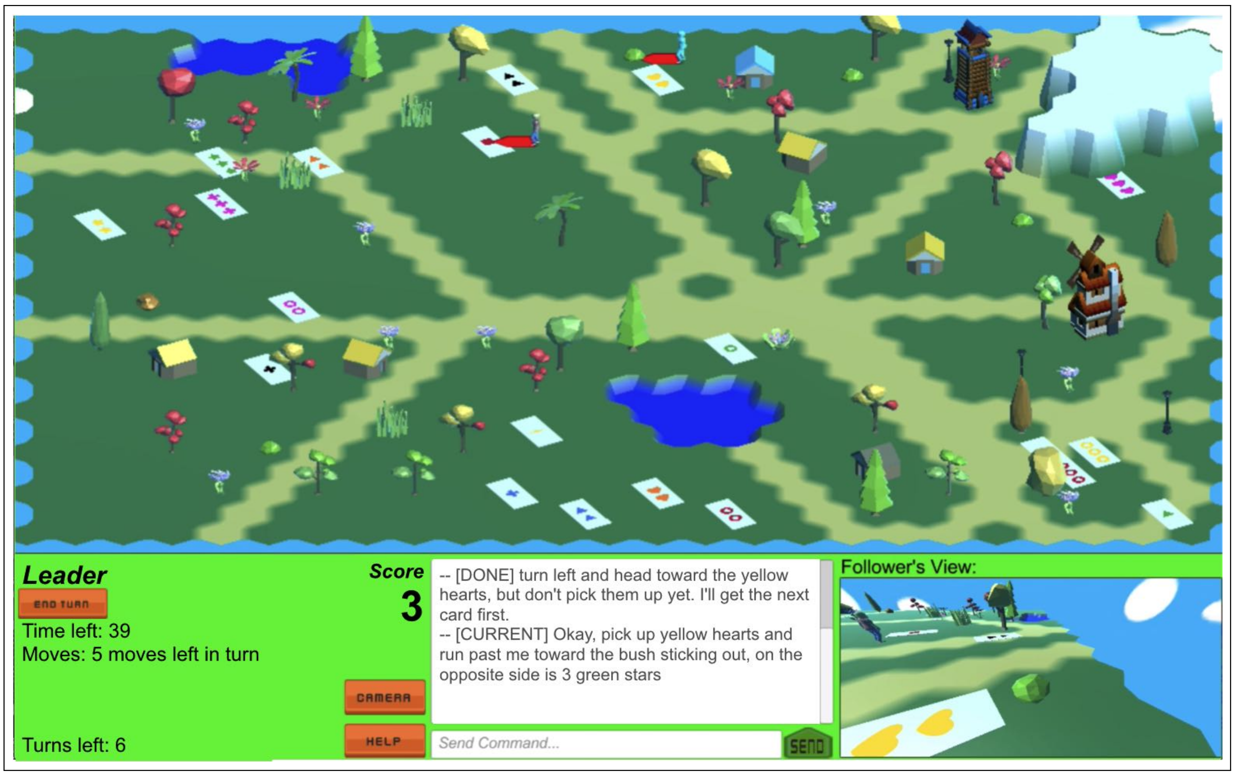
Rule 8: Both agents can take actions which modify the world state s. Each such action $a \in \mathcal{A}_w$ costs a step. We assume access to a domain-specific transition function, $\mathcal{T}_w : \mathcal{S} \times \mathcal{A}_w \rightarrow \mathcal{S}$, that describes how an environment action modifies 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.

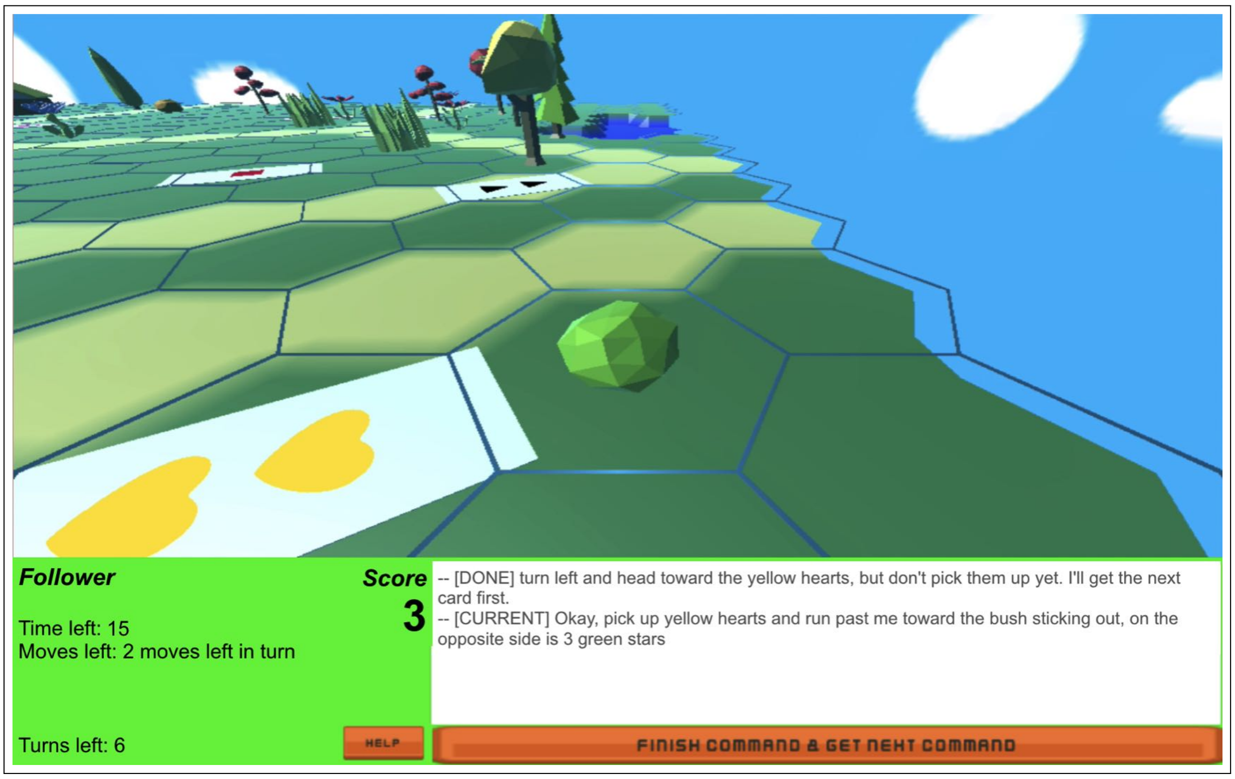
Crowdsourcing Management We use a qualification task to both teach workers how to play the game and to mark workers as qualified for our main task. We restrict those who can qualify to workers located in majority English-speaking countries with at least 90% approved HITs and at least 100 completed HITs. 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



-- [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.

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

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 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.

	Mean	Median	Max
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.

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).

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. (2018).

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:

$$\mathbf{F}_l = \text{NORM}(\text{RELU}(\text{RELU}(\mathbf{F}_{l-1} * \mathbf{K}_l^C) * \mathbf{K}_l^{C\prime})) \ .$$

We use a stride of two when convolving with $\mathbf{K}_l^{C\prime}$, 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 $\mathbf{F}_l$:

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

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

In the last stage, a series of transposed convolutions[1] are applied starting from the bottom layer and gradually synthesizing a larger feature map. For $l > 1$:

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

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 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.

[1]We use $*^\top$ to represent the transposed convolution operation.

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:

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

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:

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

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:

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

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

The auxiliary goal-prediction loss is:

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

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

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

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: $\phi^{\mathcal{X}}$, $\text{RNN}^{\mathcal{X}}$, $\phi^{\mathcal{S}}$, $\mathbf{W}_s$, $\mathbf{b}_s$, and LINGUNET. The action generation stage (Stage 2) includes the following parameters and parameterized components: CNN^P, $\mathbf{W}_1^P$, $\mathbf{W}_2^P$, $\mathbf{b}_1^P$, $\mathbf{b}_2^P$, $\phi^{\mathcal{A}}$, $\text{RNN}^{\mathcal{A}}$, $\mathbf{W}^{\mathcal{A}}$, 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.

References

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

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*.

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.

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.

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.

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*.

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.

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.

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

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*.

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*.