Continual Learning for Grounded Instruction Generation by Observing Human Following Behavior

Noriyuki Kojima, Alane Suhr, Yoav Artzi
Department of Computer Science and Cornell Tech, Cornell University, USA
nk654@cornell.edu {suhr, yoav}@cs.cornell.edu

Abstract

We study continual learning for natural language instruction generation, by observing human users' instruction execution. We focus on a collaborative scenario, where the system both acts and delegates tasks to human users using natural language. We compare user execution of generated instructions to the original system intent as an indication to the system's success communicating its intent. We show how to use this signal to improve the system's ability to generate instructions via contextual bandit learning. In interaction with real users, our system demonstrates dramatic improvements in its ability to generate language over time.

1 Introduction

Natural language provides an expressive and accessible avenue to instruct non-expert users. The ability to generate instructions is critical for systems that collaborate with users, for example, to delegate tasks. In such scenarios, the system generates language to communicate to the user a latent intent. When users are cooperative and proficient in the language, whether they accomplish the system's intent provides an informative, albeit noisy signal of the quality of instruction generation.

This implicit signal is fundamentally different from supervised data, including via active learning, in that it does not label the system's intent with a written instruction, but only provides evidence to the quality of a given instruction in relaying this intent. As a natural byproduct of interaction with users, it also differs from explicit user feedback in not requiring user action beyond what they already do as part of the interaction. Despite its potential and prevalence, this signal is understudied for learning to generate natural language

In this paper, we study this learning signal. We formalize continually improving instruction generation by observing human users executing generated instructions. We learn by comparing instruction execution to the system intent, and demonstrate how this results in a system that continually improves its natural language generation ability through interaction with users. Figure 1 illustrates our learning process.

We design a task-oriented collaborative scenario using the CerealBar game environment (Suhr et al., 2019). In CerealBar, two agents, a leader and a follower, work together to complete tasks. The leader plans the tasks to complete, and communicates goals to the follower using natural language. CerealBar was originally introduced for studying follower instruction execution. We modify it to focus on generation of leader instructions, which are then executed by human followers. The collaborative, embodied setup effectively engages users, and aligns their incentives with executing the system's instructions to the best of their abilities.

A major challenge is inferring a learning signal from observed user behavior. Given the user execution, we create positive and negative examples, depending on how the user execution aligns with the system's plan and the user's perceived correctness of their own execution. For example, consider an execution that does not align well with the system's plan, but that the user considers correct given the instruction. Because of the misalignment, we cannot consider the instruction as a successful example given the system's plan. However, given the user's perceived correctness, we can generate a positive example treating the user's execution as a plan paired with the instruction. In contrast to supervised learning with gold-standard per-token labels (Sutskever et al., 2014), such utterance-level binary labels form a challenging signal for learning, because they do not distinguish between correct and incorrect tokens.

We do not make the typical distinction between training and deployment; as human users follow

Transactions of the Association for Computational Linguistics, vol. 9, pp. 1303–1319, 2021. https://doi.org/10.1162/tacl_a_00428
Action Editor: Andreas Vlachos. Submission batch: 5/2021; Revision batch: 8/2021; Published 12/2021.

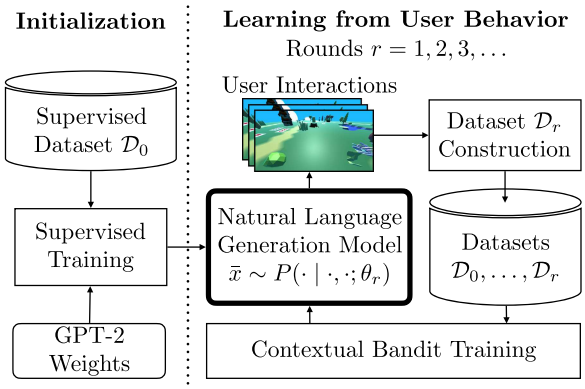


Figure 1: Diagram of our learning process. We initialize a generation model using supervised learning, and continually learn through interaction with users, by alternating between observing user execution of generated instructions and training.

generated instructions, we continually collect new data, periodically train using this data, and evaluate the system through the interaction itself. We formalize learning as an off-policy contextual bandit learning problem. We show that positive examples can be treated in a manner that reduces to supervised learning, allowing for simple effective use of the data. However, using negative examples is more challenging, because simply minimizing their likelihood gives an unbounded negative loss. We weigh negative examples using an inverse propensity score (IPS; Horvitz and Thompson, 1952; Wang et al., 2017) to address this issue.

We experiment with our approach through interaction with human users, tracking both task performance and how the generated language changes. We observe dramatic improvements in the quality of instructions generated as reflected in users' execution: Task completion in accordance to the system intent increases from 44.7% to 79.3%. This is accompanied by significant language change: The occurrence of erroneous phrases decreases as desired, but the effective system vocabulary gradually shrinks.

Although using user feedback for improving language generation has been studied, as we discuss in Section 8, to the best of our knowledge, this study is the first to show effective instruction generation learning by observing user execution. Our experiments demonstrate the effectiveness of our process, but also illustrate limitations and important directions for future work. Code and data are available at `https://lil.nlp.cornell.edu/cerealbar/`.

2 Technical Overview and Notation

Our goal is to continually improve a natural language instruction generation model, by observing human executions of generated instructions.

Interaction Scenario We focus on a collaborative scenario, where two agents, a leader and a follower, complete tasks in an environment. The system is the leader, and the human user is the follower. The leader plans tasks to accomplish, acts in the world, and instructs the follower using natural language. We use a deterministic procedure for planning and executing leader actions, and focus on learning the leader instruction generation model. The human follower acts in the world following the system instructions. We instantiate this scenario using CerealBar (Section 3), a collaborative game, where two agents collect sets of cards together by moving in a 3D environment.

Task A world state s describes the current environment; in CerealBar, this includes the location of landmarks, cards, and both agents. A plan $\bar{p}$ is a sequence of poses $\langle p_1, \ldots, p_{|\bar{p}|} \rangle$ the system intends for the human user to take starting from a start state s_1. In CerealBar, a plan includes moving in the environment with the intent of collecting cards; each pose p_j is a tuple (h_j, w_j, α_j), where h_j and w_j are height and width coordinates, and α_j is a discrete orientation angle. An instruction $\bar{x}$ is a sequence of tokens $\langle x_1, \ldots, x_{|\bar{x}|} \rangle$. An instruction execution $\bar{e}$ is the sequence of poses $\langle p_1, \ldots, p_{|\bar{e}|} \rangle$ a user takes executing $\bar{x}$, starting in a start state s_1. The generation distribution $P(\bar{x} \mid s_1, \bar{p}; \theta)$ is parameterized by θ. The goal of instruction generation is that given a generated instruction $\bar{x} \sim P(\cdot \mid s_1, \bar{p}; \theta)$, the user execution $\bar{e}$ from s_1 will follow the plan $\bar{p}$. The user does not have access to $\bar{p}$, but only to its description $\bar{x}$.

Learning We use an encoder-decoder neural network model (Section 4), which we continually improve by observing user behavior. This process proceeds in rounds. At each round r, we first collect data and then train our model by estimating the model parameters θ_r. During data collection in round r, we sample from our model to generate instructions, and observe a human user's execution of each instruction. An execution of an instruction $\bar{x} \sim P(\cdot \mid s_1, \bar{p}; \theta_r)$ generated for the plan $\bar{p}$ with start state s_1 creates a tuple $(s_1, \bar{p}, \bar{x}, \bar{e}, f)$, where $\bar{e}$

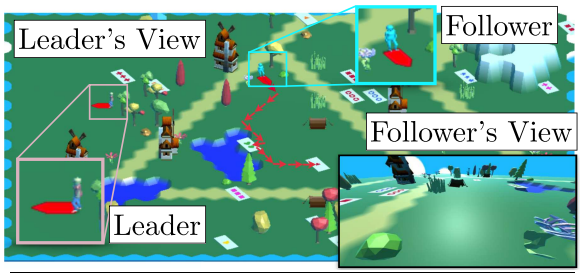
is the user execution and f is structured user feedback solicited using binary questions (e.g., about the grammaticality of $\bar{x}$). The learner does not observe the user's actions executing $\bar{x}$, but only their poses along the execution. Given these tuples, we create a dataset $\mathcal{D}_r = \{(s_1^{(i)}, \bar{\rho}^{(i)}, \bar{x}^{(i)}, y^{(i)})\}_{i=1}^{|\mathcal{D}_r|}$, where $y^{(i)} \in \{-1, +1\}$ is a binary label. Depending on the user execution and feedback, the plan $\bar{\rho}^{(i)}$ is either the original plan $\bar{p}^{(i)}$ used for generating $\bar{x}^{(i)}$ or the user execution $\bar{e}^{(i)}$ of $\bar{x}^{(i)}$. We formulate estimating θ_{r+1} as a contextual bandit learning problem with y as the reward. Section 5 describes the complete learning process.

Evaluation Throughout the system's lifetime, we measure how well human users complete tasks, and also use earth mover's distance (EMD; Rubner et al., 1998) to quantify the similarity of the user execution $\bar{e}$ to the plan $\bar{p}$. We characterize language change over time by tracking vocabulary size, instruction length, and other statistics.

3 Interaction Scenario

Suhr et al. (2019) describe CEREALBAR in detail. CEREALBAR is a two-player, turn-based game where a leader and follower collaborate to collect sets of matching cards. The game objective is to collect as many valid sets as possible in a 3D environment. The environment includes landmarks (houses, mountains, ponds, etc.) that the players must move around, and may obscure a player's view. A valid set consists of three cards with three distinct colors, shapes, and counts. Players move onto cards to select or deselect them. When the selected cards comprise a valid set, the players earn a point, all cards disappear,[1] and new cards appear. The two players must collaborate effectively using natural language. The leader observes the entire environment, plans who should select which cards for the next set, executes their own part of this plan, and issues instructions to the follower. The follower executes leader instructions, only seeing a partial first-person view of the environment. Leader instructions must make use of the observed spatial environment, including landmarks, for the follower to be able to execute them given their partial view. Each interaction includes multiple instructions. Figure 2 shows the game and example generated instructions.

[1]In Suhr et al. (2019), only the selected cards disappear. We introduced this modification to minimize inter-turn effects for the follower (i.e., memorize card locations).



...
$\bar{x}_8$: *get the two orange diamonds and the one green plus. then turn around and grab the three black hearts*
$\bar{x}_9$: ***go around the lake and get the 2 green hearts and the 1 red triangle***
`[Set made. New score: 6]`
...

Figure 2: Interaction snapshot in CEREALBAR, with instructions generated by our model. The current instruction is $\bar{x}_9$. The leader plan is illustrated with red arrows in the leader's view. The user sees only the follower's view during execution.

CEREALBAR was originally used for learning a follower instruction execution model from human demonstrations (Suhr et al., 2019). In contrast, we learn an instruction generation model for the leader, with the human user as the follower. The generated instructions must often specify multiple tasks to complete (i.e., when the follower is to select multiple cards), and how to navigate to the target cards, because the follower has only partial observability of the environment. This includes references to landmarks, spatial relations, and descriptions of paths. We focus on language generation, and use a deterministic planner to generate the plan, including which cards to select and how each player should move in their next turn, and execute the planned leader actions. The system uses the model we learn to map the follower's part of the plan to a natural language instruction.

We learn through interactions with non-expert human followers, which CEREALBAR is particularly suited for. The utility-maximizing game objective to earn a high score by collecting as many valid sets as possible incentivizes followers to execute the generated instructions as accurately as possible. In addition, CEREALBAR players need no expert knowledge to participate in the game, beyond familiarity with the simple game rules.

4 Model

We design a relatively simple encoder-decoder architecture to model the generation distribution

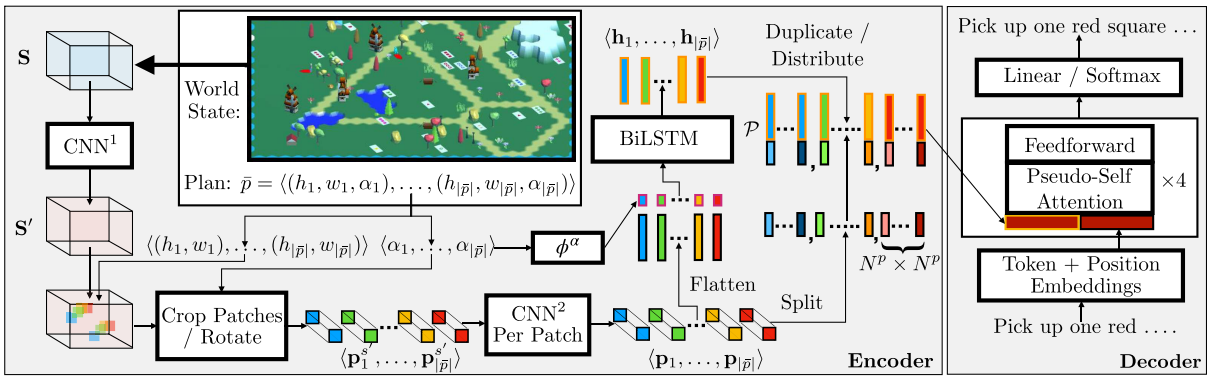


Figure 3: Model illustration. Section 4 describes the model.

$P(\cdot \mid s_1, \bar{p}; \theta)$, leaving more complex model development for future work. The inputs are a start state s_1 and a plan $\bar{p}$. The model parameters are θ. Our design considers the environment and plan to generate relevant, grounded instructions. Figure 3 illustrates the model.

Inputs Similar to Suhr et al. (2019), we represent the world state $s_1 \in \{0,1\}^{P \times H \times W}$ as a binary 3D tensor, where P is the number of position properties, and H and W are the environment's height and width. Each of the $W \times H$ positions is represented as a binary properties vector of length P (encoding the type of object in the position, its color, etc.). The system plan $\bar{p} = \langle p_1, \ldots, p_{|\bar{p}|} \rangle$ is a sequence of follower poses along the intended execution. Each pose p_j is a tuple (h_j, w_j, α_j) of height h_j and width w_j coordinates, and a discrete orientation angle α_j.

Encoder The encoder computes a set of hidden states, which the decoder attends to during generation. We use a learned embedding function ϕ^s to map each position vector to a dense embedding of size N^s by summing the embeddings of each of the position's properties. We combine the embeddings into a tensor $\mathbf{S} \in \mathbb{R}^{N^s \times H \times W}$, and compute: $\mathbf{S}' = \text{CNN}^1(\mathbf{S})$, where CNN^1 is a learned convolution and $\mathbf{S}' \in \mathbb{R}^{N^{s'} \times H \times W}$. Because the CerealBar environment is a grid of hexagons, we use HexaConv (Hoogeboom et al., 2018). We encode the plan positions into a sequence of vectors $\langle \mathbf{p}_1^{s'}, \ldots, \mathbf{p}_{|\bar{p}|}^{s'} \rangle$ by cropping a $N^{s'} \times N^p \times N^p$-sized tensors from $\mathbf{S}'$ centered around each (h_j, w_j) and rotated by α_j. These tensors represent the pose of the follower and its surroundings during execution. Each $\mathbf{p}_j^{s'}$ is encoded to $\mathbf{p}_j = \text{CNN}^2(\mathbf{p}_j^{s'})$, while retaining the dimensionality of $\mathbf{p}_j^{s'}$.

We concatenate an orientation embedding $\phi^\alpha(\alpha_j)$ to each $\mathbf{p}_j$, and process $[\mathbf{p}_1; \phi^\alpha(\alpha_1)], \ldots, [\mathbf{p}_{|\bar{p}|}; \phi^\alpha(\alpha_{|\bar{p}|})]$ with a bidirectional LSTM to compute $\mathbf{h}_1, \ldots, \mathbf{h}_{|\bar{p}|}$. We construct the set of hidden states $\mathcal{P}$ the decoder attends to by concatenating each $\mathbf{h}_j$ with the $N^p \times N^p$ position vectors encoded in each $\mathbf{p}_j$:

$$\mathcal{P} = \{ [\mathbf{h}_j; \mathbf{p}_j[x,y]] \mid 1 \le j \le |\bar{p}|, \; 1 \le x, y \le N^p \} \quad (1)$$

where $\mathbf{p}_j[x,y]$ is a position vector of size $N^{s'}$.

Decoder The decoder computes a probability distribution over token types conditioned on the prefix generated so far and the set $\mathcal{P}$, which represents the environment state and plan. The decoder uses the first four layers of the GPT-2 Transformer architecture (Radford et al., 2019), which enables initializing with GPT-2 weights. We extend it with pseudo self attention (Ziegler et al., 2019) to condition the generation on the encoder outputs $\mathcal{P}$. This adds a linear layer that projects the encoder outputs $\mathcal{P}$ into the decoder self-attention space.

Inference We decode instructions from $P(\cdot \mid s_1, \bar{p}; \theta)$ using temperature sampling with a temperature of τ (Kreutzer et al., 2018b). This sharpens the sampling distribution, to focus on higher probability outputs. We do not use beam search.

5 Learning

We continually improve our model by observing users following generated instructions and re-estimating the model parameters. We initialize

the model parameters θ_1 using an existing language model and training on a static dataset of instructions $\mathcal{D}_0$ (Section 5.1). We then perform a series of rounds, each round r includes deploying the model with human users and training on the collected interactions (Section 5.2). In round r, we collect interactions between our model parameterized by θ_r and human followers, to create a dataset $\mathcal{D}_r = \{(s_1^{(i)}, \bar{\rho}^{(i)}, \bar{x}^{(i)}, y^{(i)})\}_{i=1}^{|\mathcal{D}_r|}$ of start states $s_1^{(i)}$, plans $\bar{\rho}^{(i)}$, instructions $\bar{x}^{(i)}$, and binary labels $y^{(i)}$. We estimate θ_{r+1} using all data collected so far $\cup_{q=0}^{r} \mathcal{D}_q$. Figure 1 illustrates our learning process.

5.1 Initialization

User interaction requires some level of minimal performance. Pilot experiments showed that a poorly initialized system is likely to frustrate users, who in turn provide little learning signal. Our initialization provides a sufficient level of grammaticality and plausibility to support user interaction, and thereby further learning.

We initialize the decoder weights with the first four layers of GPT-2 (Radford et al., 2019). All other weights, including of the encoder and pseudo self-attention linear layers, are initialized randomly. We then train with a supervised dataset $\mathcal{D}_0 = \{(s^{(i)}, \bar{\rho}^{(i)}, \bar{x}^{(i)}, y^{(i)})\}_{i=1}^{|\mathcal{D}_0|}$ of human plans $\bar{\rho}^{(i)}$ starting at start states $s^{(i)}$ and instructions $\bar{x}^{(i)}$, all with positive labels $y^{(i)} = +1$. We use limited data, just sufficient to effectively interact with users for further learning. We estimate θ_1 by minimizing a supervised loss:

$$\mathcal{L}_I(\theta_1, \mathcal{D}_0) = - \frac{1}{|\mathcal{D}_0|} \sum_{i=1}^{|\mathcal{D}_0|} \log P(\bar{x}^{(i)} | s^{(i)}, \bar{\rho}^{(i)}; \theta_1) \; . \quad (2)$$

5.2 Learning from User Behavior

Learning from interacting with human users alternates between generating instructions in interaction with users and training the model.

Interaction with Users In each round r, we first deploy the model with parameters θ_r to interact with human users, with our system as the leader and the user as the follower. We do not update the model during this interaction phase.

The game environment is randomly generated for each interaction. Each game continues until it concludes, either when the user leaves or the turns are exhausted. A game often includes collecting multiple sets of cards, and generating multiple instructions. Each instruction is generated for the current state as the start state s_1;[2] as both agents move and change the status of cards, the environment state changes throughout the game. At state s_1, we generate the plan $\bar{p}$ using a deterministic planner that determines (a) which cards should be selected or de-selected to make the next valid set, and (b) the shortest paths the leader and follower should take to visit all target cards. The actions the planner assigns to the follower form the plan $\bar{p}$. The actions assigned to the leader are executed by the leader agent deterministically during its turn. The model is used to sample an instruction $\bar{x} \sim P(\cdot \mid s_1, \bar{p}; \theta_r)$, which is displayed to the user. The human user has no access to $\bar{p}$, the set of target cards, or the game state s_1. They only observe the instruction and what is ahead (Figure 2).

During their turn, the user executes $\bar{x}$ to the best of their ability, and indicates when done. If the user determines that the instruction cannot be followed, they can terminate the execution, which is treated just like marking the instruction as complete. The user execution $\bar{e}$ is the entire sequence of poses they take while following the instruction.

When the user concludes or terminates an instruction $\bar{x}$, we show them a top-down view of the entire environment with their execution path highlighted. They do not see the original system plan. We ask the user two binary feedback questions about the perceived correctness of their execution and grammaticality (Figure 4).

We create a tuple $(s_1, \bar{p}, \bar{x}, \bar{e}, f)$ for each execution $\bar{e}$, where s_1 is the start state of the environment, $\bar{p}$ is the plan generated in that state, $\bar{x} \sim P(\cdot \mid s_1, \bar{p}; \theta_r)$ is the sampled instruction, and f is the set of responses to the feedback questions.

Perceived correctness: *Did you follow all parts of the Leader's command and find everything correct?*
Users are instructed to answer *yes* only if they consider their execution correct given the instruction, if they perceive that the instruction describes the relevant cards and objects correctly, and if the specified actions make sense.

Grammaticality: *Was the instruction grammatical and well written?*
Users are shown examples of errors before the game, and largely interpret this criteria as language correctness independent of the world state.

Figure 4: The binary questions displayed to the user at the end of instruction execution.

[2] For simplicity, we do not index the game time step.

Once the user submits the answers to the feedback questions, the next instruction is generated.

Dataset Construction We use all interactions in round r to construct dataset $\mathcal{D}_r$, which is made of tuples $(s_1, \bar{\rho}, \bar{x}, y)$, where $\bar{\rho}$ is a plan and y is a binary label. Given a tuple $(s_1, \bar{p}, \bar{x}, \bar{e}, f)$, we use three heuristics to add examples to $\mathcal{D}_r$:

1. If any feedback answer in f is negative, the instruction does not reflect the user's execution or not well written (i.e., ungrammatical). We add a negative example to $\mathcal{D}_r$ with the system plan $\bar{p}$: $(s_1, \bar{p}, \bar{x}, -1)$.

2. If both feedback answers are positive, the user considers their execution $\bar{e}$ accurate and the instruction well formed. This does not necessarily indicate the execution follows the system plan, but that we can treat the execution as a plan. We add a positive example with the execution as the plan: $(s_1, \bar{e}, \bar{x}, +1)$.

3. If both answers are positive and the execution $\bar{e}$ follows the plan $\bar{p}$,[3] the instruction communicates the plan well. We add a positive example with the system plan: $(s_1, \bar{p}, \bar{x}, +1)$.

Overall, we add examples to $\mathcal{D}_r$ using both the original system plan and the user execution. The heuristics utilize the observational learning signal as much as possible while avoiding examples not beneficial for learning. For example, we do not add negative examples using the user execution, because these are less likely to be useful for learning. Although such executions can form negative examples if the user answered negatively to the correctness question, they tend to be relatively arbitrary, and it is unlikely the model conditioned on them will assign significant probability to the generated instruction, which is the behavior negative examples come to suppress.

Parameter Estimation We estimate the model parameters for the next round θ_{r+1} using all available data $\mathcal{D} = \cup_{q=0}^{r} \mathcal{D}_q$. We re-train our model, starting with GPT-2 parameters (Section 5.1).[4]

We formulate learning as an offline contextual bandit problem, treating the sentence labels y as rewards. Learning from the positive examples in $\mathcal{D}$ forms a straightforward supervised learning problem, albeit one where the data is generated from system interaction. A key challenge is using the negative examples. Treating them like supervised examples requires optimizing the probability of their instructions to zero. Because $\lim_{P(\cdot)\to 0} \log P(\cdot) = -\infty$, this leads to an unbounded negative loss that quickly dominates the objective. This in contrast to positive examples, for which the loss is bounded by zero. This issue is not present in existing work using offline contextual bandits to improve machine translation (Lawrence et al., 2017; Kreutzer et al., 2018b), where rewards are always non-negative.

We address this issue by adding an inverse propensity score (IPS; Horvitz and Thompson, 1952; Wang et al., 2017) coefficient to negative examples in a policy gradient objective. The gradient for estimating parameters θ_{r+1} is:

$$\nabla \mathcal{L}(\theta_{r+1}, \mathcal{D}) = \frac{1}{\mathcal{D}} \sum_{i=1}^{|\mathcal{D}|} \ell_{\theta_{r+1}}^{(i)} y^{(i)} \nabla \log P(\bar{x}^{(i)} \mid s^{(i)}, \bar{\rho}^{(i)}; \theta_{r+1}) \ , \quad (3)$$

where, given an example $(s^{(i)}, \bar{\rho}^{(i)}, \bar{x}^{(i)}, y^{(i)})$ acquired in round q with parameters θ_q, $\ell_\theta^{(i)}$ is:

$$\ell_\theta^{(i)} = \begin{cases} 1 & y = +1 \\ \frac{P(\bar{x}^{(i)} | s^{(i)}, \bar{p}^{(i)}; \theta)}{P(\bar{x}^{(i)} | s^{(i)}, \bar{p}^{(i)}; \theta_q)} & y = -1 \end{cases} . \quad (4)$$

As the probability of a negative example (i.e., $y = -1$) decreases, so does its impact on the loss. While IPS is commonly used in bandit learning to de-bias the loss estimate (Lawrence et al., 2017), our motivation is different, and we do not add it to positive examples. Because of the large combinatorial space, sentence probabilities are generally small. The IPS coefficient of a positive example can become very large as its probability increases during learning. Instead, we use a supervised-like term, which is known to behave well.[5]

[3] For instructions that target cards, we require getting the card selection right, and ignore the follower position. For instructions that require waiting (e.g., *hold still*), we require the position to remain the same, but allow orientation deviation.

[4] Pilot studies showed re-training to be more stable than fine-tuning given new data, and we conduct the majority of our experiments with this method. However, we also observe that our process is overall robust to the initially observed instabilities of fine-tuning (Section 7).

[5] An alternative, and important direction for future study is to add IPS to all examples, but clip it at a certain maximal value, similar to clipping in PPO (Schulman et al., 2017).

6 Experimental Setup

Initialization Data We create the supervised initialization dataset $\mathcal{D}_0$ by sampling 360 interactions from the original CEREALBAR data (Suhr et al., 2019), which was collected in a wizard-of-oz (WOZ; Kelley, 1984) setup via human-human games. We select this number through pilot studies and qualitative analysis to minimize the amount of initialization data, while still maintaining sufficient model performance for early interactions to facilitate learning. Our goal is to use as little data as possible to study the target scenario where investment in supervised data is minimal, and most learning is left to interaction with users. This data includes 7,147 examples. We use the human demonstrations in the original data as plans.

Evaluation Similar to Zhao et al. (2021), we observe that automated metrics, such as BLEU (Papineni et al., 2002) or BERTScore (Zhang et al., 2020), computed over a static held-out validation set are unreliable for evaluating instruction generation. Instead, we focus on task-completion measures via human execution. We measure *task completion* by considering the user execution as completing the intended task if the user visits all card locations included in the system plan; or, if the plan includes no target cards, the user stays in the starting position. We quantify the similarity of the user execution to the path in the system plan by computing *earth mover's distance* (EMD; Rubner et al., 1998)[6] between the two (Blukis et al., 2019). We also track the user answers to the feedback questions (Figure 4). We average each measure over the number of instructions in each round.

Language Analysis We quantitatively analyze how generated instructions change throughout training. For each round, we report mean instruction length, vocabulary size, and three measures of syntactic complexity using dependency trees (Xu and Reitter, 2016): (a) maximum depth: the longest path from root to a leaf; (b) maximum width: the maximum out-degree of any word in the tree; and (c) average branching factor: the average out-degree of non-leaf words. We normalize the three measures by instruction length. We qualitatively analyze errors in generated instructions, by comparatively analyzing 100 randomly sampled examples where the user failed to complete the intended task from the first and final rounds.

Interaction Setup Except initialization, learning and evaluation are done through live interaction with users on Amazon MTurk. All workers passed a tutorial and a qualification quiz. We pay \$0.15 per interaction, with a bonus of \$0.10 per instruction to workers who follow our guidelines.

Implementation Details Similar to performance evaluation, automated measures are unreliable for model selection. Instead, for both initialization and in each round, we train for $N = 400$ epochs, and take the final model. We find N via qualitative analysis of the initial model. We use an ensemble of four models. We uniformly sample one of the four models to sample each instruction, and take its probability to use in IPS for negative examples. We use a sampling temperature $\tau = 0.5$, and AdamW (Loshchilov and Hutter, 2018) for learning.

7 Results and Analysis

We conduct a long-term experiment with 14 rounds using our approach, and separate seven-round experiments to compare system variants. In both experiments, we collect roughly 100 interactions for each system per round. In the seven-round experiments, we deploy methods simultaneously to ensure that our observations are not sensitive to changes in user behavior, for example, because of adaptation and increased expertise. We do not inform workers about the model they are interacting with. We train each system only on data collected by the same method in previous rounds.

7.1 Long-term Study

We experiment with our approach for 14 rounds. We collect a total of 27,031 instructions from 1,445 interactions, with 103.2 interactions per round on average. The total cost is \$2,895. Figure 5 shows both performance measures and language trends. For task measures and user feedback, we also break down performance according to the number of target cards in the system plan to evaluate performance changes for plans which may be more difficult to describe (e.g., because they require specifying more cards).[7]

[6] We use POT (Flamary et al., 2021) to compute EMD.

[7] 0-card plans target no cards (e.g., *hold still*).

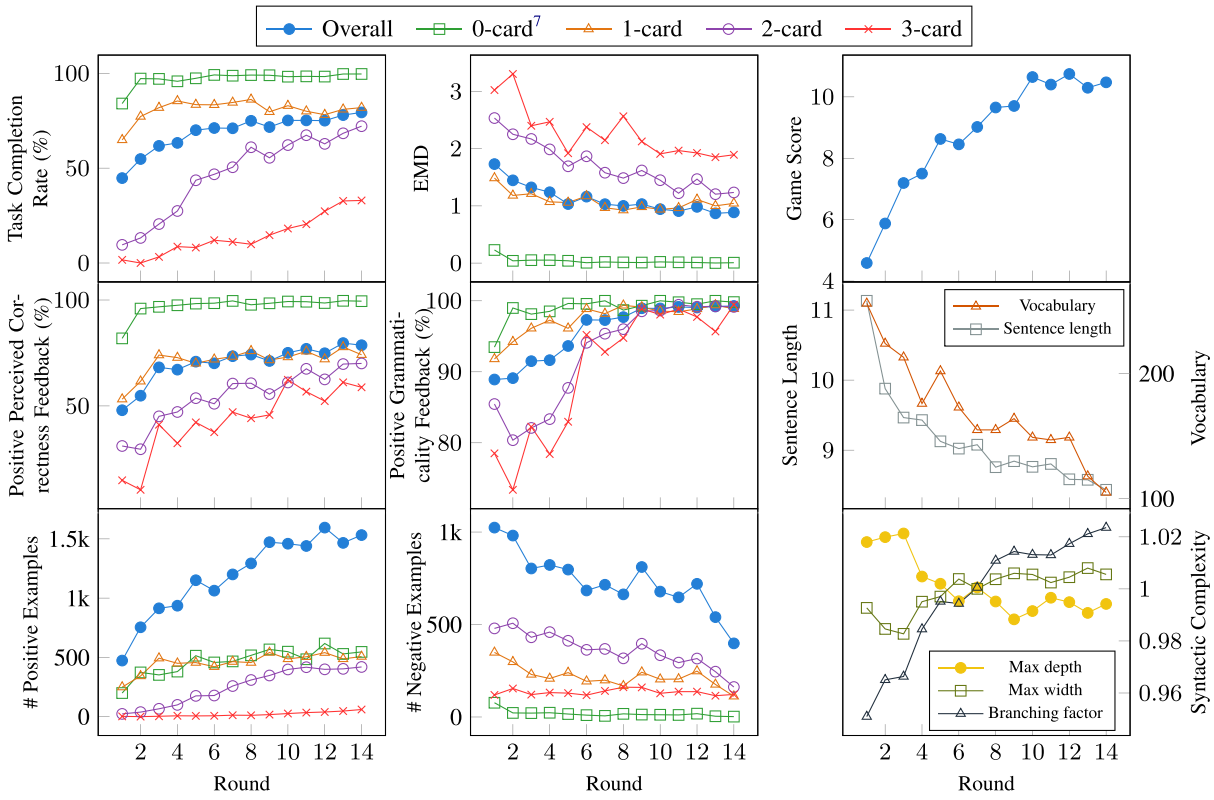


Figure 5: The system's lifetime statistics from the long-term experiment (14 rounds). The system improves on task completion (↑), EMD (↓), positive response rate for the two feedback questions (↑), and game score (↑). Section 7.1 discusses these results in detail.

Our learning method significantly improves the system performance across all measures. Task completion rate improves from 44.7% at round one to 79.3% at round 14, while EMD decreases from 1.73 to 0.88, showing increasing similarity between the execution and the plan. The user perception of the system also improves: The positive response rate for the perceived correctness question improves from 47.9% to 78.6%, and for grammaticality from 88.9% to 99.2%. The overall collaborative system performance improves as well; the game score increases from 4.5 to 10.4. The number of positive examples per round gradually increases, as the system improves and the interactions become longer. In contrast, the number of negative examples decreases over time.

We observe that the initial model struggles to describe plans containing more target cards, with a particularly low task completion rate of 1.6% for 3-card plans in the first round. This is potentially because only 0.7% of human follower executions in $\mathcal{D}_0$ demonstrate picking up three cards, while the planner generates 3-card plans 7.9% of the time. While initial improvement is slow for 3-card instructions, it picks up around round eight, and reaches 32.9% task completion rate.

Language Analysis We observe a consistent trend of decreasing sentence length and vocabulary size. Overall, these trends accompany reduction in over generation of erroneous phrases that are not grounded well in the environment. We also qualitatively observe that the systems gradually generates slightly more underspecified instructions, for example by dropping mentions of landmarks crucial for navigating to a target card. This may explain the slight decrease in 1-card task completion rate in later rounds (Figure 5), because the planner usually has the follower travel further for 1-card instructions, which requires relying more on landmarks. A potential explanation to the decrease in vocabulary size is the ever increasing presence of system-generated sentences in training, which reinforces the system's word choices. Alternatively, our learning signal may not account for the need for more descriptive language. For example, humans may compensate with exploration for omitted descriptions, which is

Error Type	$r = 1$	$r = 14$	Example
Incorrect, missing, or extra cards	75	39	*turn left and go to the yellow ~~star~~ triangles*
Irrelevant landmarks	13	1	*Head toward the ~~windmill~~ house. grab 2 red and triangle*
Incorrect direction	30	35	*grab the black heart to ~~your left~~ in front of you.*
Incorrect actions or conditions	28	14	*~~After the two red triangles~~, get the 3 red triangles.*
Underspecification	8	26	*turn right and go straight toward red trees collect two orange triangle.*
Implausible instructions	11	1	*Turn left and get the two pink hearts ~~and the two pink hearts near the pink hearts.~~*
Proportion of erroneous instructions	68.5%	26.8%	

Table 1: The types of errors observed in erroneous instructions generated during the first ($r = 1$) and final ($r = 14$) rounds of deployment. We show error counts from the 100 randomly-sampled erroneous instructions. Examples illustrate error categories; red strikethrough shows erroneous segments, and blue fragments show possible corrections. Instructions that fit into multiple categories are double counted.

not distinguished by how we convert the observed behavior to a learning signal. These trends outline important directions for future work.

We observe a small increase in syntactic complexity over the system's lifetime with regard to the branching factor, which shows significant increase ($p < 0.00001$).[8] We also see a slight decrease in maximum tree depth ($p < 0.0001$), and no significant change in max width.

Error Analysis We analyze errors in the generated instructions at the first and final rounds. For each round, we randomly sample 100 instructions that the user did not execute according to the plan or answered negatively to a feedback question. Table 1 shows error types and example instructions. Overall, the frequency of erroneous instructions decreases from 68.5% of instructions in the first round, to 26.8% in the final round. From the first to final round, we observe noticeable decrease in errors related to grounding of cards and landmarks. The overall frequency of errors related to incorrect directions and incorrect actions or conditions also decreases, and implausible instructions diminish close to zero percent. However, there is an overall increase in underspecified instructions. This aligns with the decrease in the vocabulary size and landmark use we discuss above.

Confounding Factors We identify two mechanisms unrelated to our approach that could explain the observed performance changes. We deploy two additional systems alongside our system during the final round. For each interaction, one of the three systems is randomly chosen. We do not inform the workers of the identity of the model for each interaction. First, we deploy the system following initialization during the final round to study if performance might be explained by user improvement over time. Second, because we train with a fixed number of epochs, later rounds have many more gradient updates, which may allow for better parameter estimation, even with the same amount of data. We train a system on the initialization dataset $\mathcal{D}_0$ for the same number of gradient updates as when training the final full system.

Model	r	Overall	0-card[7]	1-card	2-card	3-card
θ_1	1	44.8	84.1	64.9	9.6	1.7
θ_1	14	45.1	84.5	62.1	9.3	0.8
θ_1'	14	49.6	76.6	63.8	24.8	7.4
θ_{14}	14	**79.4**	**99.6**	**81.9**	**72.1**	**33.0**

Table 2: The effect of confounding factors on task completion rate (%). The initial model θ_1 is evaluated both in the first ($r = 1$) and final ($r = 14$) rounds, showing no effect of user adaptation. In the final round, we also evaluate θ_1', which is trained on the same data as θ_1 but using more gradient updates. We also show results for the final-round model θ_{14}.

Table 2 shows that these confounding factors do not explain the observed gains. We find minimal differences between evaluating the initial model (θ_1) at the beginning and end of deployment, showing no significant effect from user improvement. Training the initial system longer (θ_1') shows a slight overall improvement, but negligent compared to final system (θ_{14}).

[8] We use t-test ($\alpha = 0.01$) comparing rounds 1 and 14.

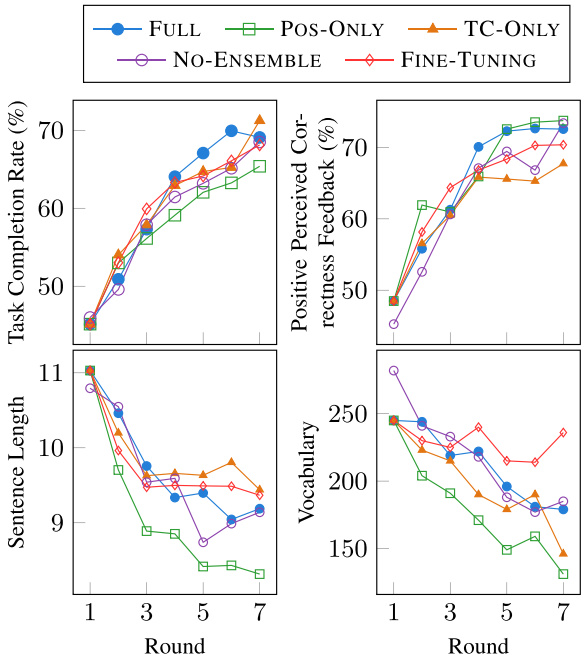


Figure 6: Comparison of system variants.

7.2 System Variants Study

We vary different design decisions, and experiment for seven interaction rounds.[9] We experiment with four system variants: (a) FULL: our full approach described in Section 5; (b) POS-ONLY: use only examples with positive labels $y = +1$; (c) TC-ONLY: ignore the feedback questions, instead if the user completes the task according to our task success measure we add positive examples with both the system plan and user execution, otherwise we add a negative example using the system plan; (d) NO-ENSEMBLE: train and deploy a single model each round, starting from an initial model randomly sampled from these we use for FULL; and (e) FINE-TUNING: train model parameters θ_{r+1} on $\mathcal{D}_r$ for N epochs, starting from θ_r, avoiding overfitting with rehearsal (Rebuffi et al., 2017; Hawkins et al., 2020a). In rehearsal, in each batch, half the examples are sampled randomly from the previous datasets $\mathcal{D}_0, \ldots, \mathcal{D}_{r-1}$. Except the variations specified, the systems are identical. We do not deploy a system ablating IPS, because we observe that training with negative examples without IPS results in a largely unusable system.

We collect a total of 63,189 instructions across all systems, with 3173 interactions. Each round includes 453.2 interactions on average. The total cost is $7,165. All systems are used concurrently in each round, including re-deploying FULL again starting from initialization. Figure 6 shows the results. Despite some differences between the system variants, our method is largely robust to variations in learning design decisions.

All systems achieve comparable improvements in task completion rate, except for OS-ONLY, which slightly underperforms. We observe faster decrease in the vocabulary size and instruction length for OS-ONLY, which does not use negative examples. This is possibly because the loss from negative examples encourages a more uniform generation distribution, potentially slowing down the overall trends of making the generation distribution more peaky. TC-ONLY, which ignores the answers to user feedback questions when constructing the dataset, shows fewer positive responses to the perceived correctness feedback, although task completion remains comparable.

We observe that using a single (NO-ENSEMBLE) model rather than an ensemble leads to limited difference in overall performance. However, because of the challenge of identifying a good automated metric to stop training, the performance of models following training varies significantly. This can lead to deploying a bad model, which provides users with a poor experience. Using an ensemble of models incurs higher computational cost, but makes such a worst-case scenario less likely. For example, in our long-term experiment, the maximum task completion performance gap we observe between the best and worst models in each round is 13%.

Finally, we observe that fine-tuning (FINE-TUNING) works as well as our re-training approach (FULL), potentially with a more stable vocabulary size. This is in contrast to our initial experiments, which showed it is harder to get consistent improvements through fine-tuning. While the fine-tuning process is harder to design because it requires to choose the fine-tuning procedure (e.g., rehearsal (Robins, 1995) or KL regularization (Yu et al., 2013)) and carefully optimize additional hyperparameters, it can work just as well as re-training. Because fine-tuning is faster to train between rounds, it may be preferable in future work.

[9] This study is similar to ablation analysis, but aims to study different learning design decisions. Full-fledged repetitive ablations to identify the ideal system design are particularly challenging in this work, both because of experiment costs and the complex dynamics of interacting with users.

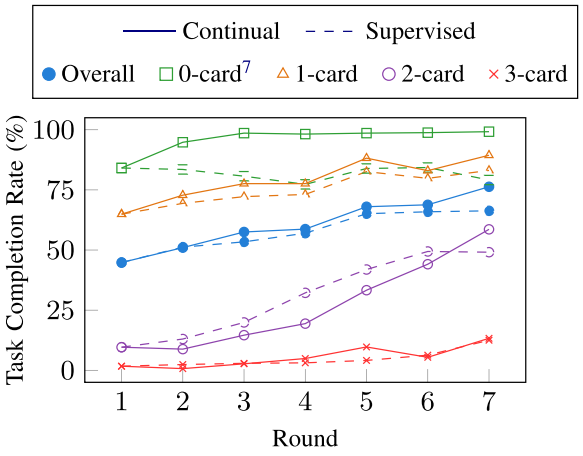


Figure 7: Comparison to supervised learning. The continual learning system is competitive in task completion rates with systems trained on equivalent amount of supervised data.

7.3 Comparison to Supervised Learning

We also separately study the learning trends of our method compared to training on equivalent amount of supervised WOZ data. Supervised data is fundamentally different from our bandit data, for two main reasons: (a) it is significantly costlier because it requires a dedicated instruction-writing effort, whereas our data arises naturally from the system interaction with users during deployment; and (b) it provides per-token labels, whereas our data includes only utterance-level binary labels. For the supervised system, after each round, we expand the dataset by randomly drawing an equivalent amount of additional data from the complete dataset of Suhr et al. (2019), which includes 19,112 examples from 960 interactions.[10] This dataset allows for seven rounds. We concurrently deploy a no-ensemble variant of our continual learning system. We collect a total of 22,216 instructions across both systems, with 1,166 interactions. This experiment's total cost is $2,230.

Figure 7 shows our continual learning system consistently outperforms this supervised alternative in overall task completion rate. There are two potential explanations to this gap. First, the data our approach uses is made of examples the system is likely to generate, potentially providing a more effective learning signal. Second, there is a difference between the plans of human leaders and our planner. Our training is better suited to adapt to how the complete system is designed, whereas training on human-annotated data is bound to suffer from a distribution shift. However, the continual learning system did not consistently outperform the supervised alternative on 2-card and 3-card instructions, especially at early rounds. This is likely because the continual learning system generates few positive examples for more complex system plans (i.e., 2-card or 3-card) at earlier rounds. At later rounds, as the system improves, we observe more positive examples for such plans, creating an accelerating effect of improvement, which is best observed in our long-term experiment (Figure 5).

[10]Interactions with the supervised system are not used for learning, but only for evaluation.

8 Related Work

Learning for instruction generation has been studied using supervised methods, with examples of task specifications (i.e., contexts) paired with human-written instructions (e.g., Daniele et al., 2016; Narayan-Chen et al., 2019), including to improve instruction following (Fried et al., 2018; Tan et al., 2019). We focus on continually learning by observing users executing generated instructions. This reduces annotation needs, and delegates much of the learning to interaction with users during system deployment. Language generation in context was also studied in scenarios that are not explicitly instructional, but aim to elicit specific behavior, such as negotiation games (e.g., Lewis et al., 2017) and referring expression generation (e.g., Dale and Reiter, 1995).

Gatt and Krahmer (2017) survey existing work on language generation, including using rule-based methods. Similar to our approach, some rule-based methods were evaluated with human followers in situated environments using task success (e.g., Koller et al., 2010; Janarthanam and Lemon, 2011). Such methods are accurate and reliable, but are limited to pre-specified rules and remain static following development. Our focus is on studying the potential for learning by observing human behavior. The two approaches can be combined, for example by using rule-based methods to generate initialization data for our approach.

Bandit learning has been studied with simulated user ratings for machine translation (Nguyen et al., 2017; Lawrence et al., 2017; Kreutzer et al., 2017) and semantic parsing (Lawrence and

Riezler, 2018). We learn from real users, similar to recent studies in machine translation (Kreutzer et al., 2018a,b). In general, such learning assumes users can judge the system output, for example via proficiency in the language they wish to translate to. Our learning signal does not require such expertise, and is available naturally from the interaction.

Explicit human feedback has also been incorporated into reinforcement learning methods (Knox and Stone, 2009; Pilarski et al., 2011; Daniel et al., 2015; Mathewson and Pilarski, 2016; Warnell et al., 2018; MacGlashan et al., 2017; Arumugam et al., 2019), including in the context of dialogue system learning (Liu et al., 2018). Jaques et al. (2020) study forming a reward from implicit feedback for non-task-oriented dialogue language generation, by training multiple models to detect linguistic signals, such as sentiment and lexical overlap, that correlate with explicit user feedback. Learning from users has also been studied by asking users to rank system outputs (e.g., Wilson et al., 2012; Christiano et al., 2017), including for instruction following (Wang et al., 2016) and summarization (Stiennon et al., 2020). Unlike our approach, such ranking requires knowing the true system intent, and is not part of the system's normal operation (i.e., instructing users in our case).

Incorporating human users into learning is related to active learning (Settles, 2009), where a policy selects examples for an oracle to label during learning. Unlike common active learning scenarios we do not select examples from a static underlying distribution (i.e., a training set) for annotation, but generate examples with the learned model. This is similar to query synthesis active learning (Angluin, 1988), where examples are generated for annotation, rather than being selected from a set of unannotated examples. A more significant difference is that active learning methods solicit model output annotations by presenting an oracle with model inputs. In contrast, our approach exposes users to model outputs (i.e., generated instructions). It does not solicit written instructions, as would be expected if requesting labels. We also do not show model inputs (i.e., plans) to users. Finally, our model interacts with users during system operation, while completing its task. It does not require oracle annotators.

Language learning from behavioral signals has been studied in the cognitive science and psychology literature.[11] Krauss and Weinheimer (1966) study two types of feedback in human studies: concurrent linguistic feedback and behavioral intent confirmation, and show how both influence linguistic adaptation in an interaction over time. Studies of reference games reproduced the effect of confirmation feedback, showing that successful intent communication reinforces convention formation in the form of shorter references (Clark and Wilkes-Gibbs, 1986; Hawkins et al., 2020b). Our learning signal is a type of confirmation feedback. However, our interaction procures and makes use of more complex learning signals than a simple binary intent communication success, by using the path the listener takes in response to the generated instruction as an alternative intent when constructing data for learning (Section 5.2).[12]

9 Discussion

We propose a methodology to continually improve an instruction generation model by observing human users executing natural language instructions, and demonstrate its efficacy within a collaborative instruction following scenario. Our study shows that observation of user behavior is an informative signal for generating language to relay instructional intent. To the best of our knowledge, this type of learning signal has not been studied before. This learning setting facilitates continual learning through interaction with users, and is particularly compelling for interactions with collaborative agents, including robots and software agents. Such agents are likely to operate in constantly changing environments (e.g., robots in homes), where continual learning is necessary to adjust to changes. Our continual learning approach also provides systems the flexibility to co-adapt to human users, who are likely to change preferences and behaviors in response to system behavior.

Our experiments demonstrate the learning process is robust to various learning and process design choices. However, they also show it is accompanied by a reduction of language complexity, including reducing the effective vocabulary and sentence length. While much of the decrease in the effective vocabulary size throughout the system

[11] This review is not comprehensive, and only aims to highlight the relation to problems studied in related disciplines.

[12] In more recent reference games (Hawkins et al., 2020b), unlike in Krauss and Weinheimer (1966), the choice of a bad referent can be seen as related to our use of listener execution.

lifetime relates to generating fewer erroneous phrases, it also reduces the language diversity and descriptiveness. Our experiments show that this trend can be slowed down by using negative examples, and appears to be less pronounced when using fine-tuning. The combination of this decrease with the preference for shorter instructions makes it difficult for the system to describe longer, complex trajectories. Qualitatively, we observe this open problem is responsible for a significant portion of the remaining errors. An important direction for future work is experimenting with directly encouraging more diverse language. This can be combined with approaches that allow for introducing new word types, which is unlikely in our approach, even though it uses sub-word tokenization. A potential direction in this vein is combining active learning to solicit human-written oracle instructions for plans the system fails to communicate.

Our work highlights several other directions for future work. There is a strong need for a reliable automated metric to evaluate instruction generation. In absence of such a metric, we use a simple, but likely sub-optimal stopping criteria for learning. Beyond the learning signal we explored in our experiments, there are additional potential cues available during interaction. For example, using continuous-valued similarity between system intent and user execution, modeling follower quality to discount the learning signal from interactions with bad followers, or weighing the feedback questions differently for more nuanced reward.

Finally, the decrease in utterance length and vocabulary size mirrors similar trends observed in studies of human communication (Clark and Wilkes-Gibbs, 1986; Hawkins et al., 2020b). This illustrates the potential of continual learning systems to reflect the dynamics of language change human participants expect in natural language interactions. Observations of human learning also indicate the potential of integrating our approach with conversational self-repair (Clark, 2020) and partner reformulation (Clark, 2018), both important components of child language acquisition that likely provide better credit assignment for learning compared to our binary bandit signal.

Acknowledgments

This research was supported by ARO W911NF-21-1-0106, a Google Focused Award, the Masason Foundation, a Facebook Fellowship, and NSF under grants no. 1750499 and DGE-1650441. We thank Jonathan Chang, Sasha Rush, the Cornell NLP Group, Robert Hawkins, Dipendra Misra, and John Langford for discussion and comments; Suyi Diao for Unity development; Anna Effenberger for code to compute syntax complexity; Ge Gao, Koji Shiono, and Takayuki Kojima for feedback on our interaction platform; and the crowdsourcing workers for participating in our data collection. Finally, we thank the action editor and the anonymous reviewers for detailed comments.

References

D. Angluin. 1988. Queries and concept learning. *Machine Learning*, 2:319–342. https://doi.org/10.1023/A:1022821128753, https://doi.org/10.1007/BF00116828

Dilip Arumugam, Jun Ki Lee, Sophie Saskin, and Michael L. Littman. 2019. Deep reinforcement learning from policy-dependent human feedback. *CoRR*, abs/1902.04257.

Valts Blukis, Eyvind Niklasson, Ross A. Knepper, and Yoav Artzi. 2019. Learning to map natural language instructions to physical quadcopter control using simulated flight. In *Proceedings of the Conference on Robot Learning*, pages 1415–1438.

Paul Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. In *Proceedings of the Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Eve V. Clark. 2018. Conversation and language acquisition: A pragmatic approach. *Language Learning and Development*, 14:170–185. https://doi.org/10.1080/15475441.2017.1340843

Eve V. Clark. 2020. Conversational repair and the acquisition of language. *Discourse Processes*, 57:441–459. https://doi.org/10.1080/0163853X.2020.1719795

Herbert H. Clark and Deanna Wilkes-Gibbs. 1986. Referring as a collaborative process. *Cognition*,

22(1):1–39. https://doi.org/10.1016/0010-0277(86)90010-7

Robert Dale and Ehud Reiter. 1995. Computational interpretations of the gricean maxims in the generation of referring expressions. *Cognitive Science*, 19:233–263. https://doi.org/10.1207/s15516709cog1902_3

Christian Daniel, Oliver Kroemer, M. Viering, Jan Metz, and Jan Peters. 2015. Active reward learning with a novel acquisition function. *Autonomous Robots*, 39:389–405. https://doi.org/10.1007/s10514-015-9454-z

Andrea F. Daniele, Mohit Bansal, and Matthew R. Walter. 2016. Natural language generation in the context of providing indoor route instructions. In *Proceedings of the Robotics: Science and Systems Workshop on Model Learning for Human-Robot Communication*.

Rémi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Z. Alaya, Aurélie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, Léo Gautheron, Nathalie T. H. Gayraud, Hicham Janati, Alain Rakotomamonjy, Ievgen Redko, Antoine Rolet, Antony Schutz, Vivien Seguy, Danica J. Sutherland, Romain Tavenard, Alexander Tong, and Titouan Vayer. 2021. POT: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8.

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. https://doi.org/10.18653/v1/N18-1177

Albert Gatt and Emiel Krahmer. 2017. Survey of the state of the art in natural language generation: Core tasks, applications and evaluation. *Journal Artificial Intelligence Research*, 61:65–170. https://doi.org/10.1613/jair.5477

Robert Hawkins, Minae Kwon, Dorsa Sadigh, and Noah Goodman. 2020a. Continual adaptation for efficient machine communication. In *Proceedings of the Conference on Computational Natural Language Learning*, pages 408–419. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.conll-1.33

Robert D. Hawkins, Michael C. Frank, and Noah D. Goodman. 2020b. Characterizing the dynamics of learning in repeated reference games. *Cognitive Science*, 44(6):e12845. https://doi.org/10.1111/cogs.12845, PubMed: 32496603

Emiel Hoogeboom, Jorn W. T. Peters, Taco S. Cohen, and Max Welling. 2018. Hexaconv. In *Proceedings of the International Conference on Learning Representations*.

Daniel G. Horvitz and Donovan J. Thompson. 1952. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47(260):663–685. https://doi.org/10.1080/01621459.1952.10483446

Srini Janarthanam and Oliver Lemon. 2011. The GRUVE challenge: Generating routes under uncertainty in virtual environments. In *Proceedings of the European Workshop on Natural Language Generation*, pages 208–211. Association for Computational Linguistics.

Natasha Jaques, Judy Hanwen Shen, Asma Ghandeharioun, Craig Ferguson, Agata Lapedriza, Noah Jones, Shixiang Gu, and Rosalind Picard. 2020. Human-centric dialog training via offline reinforcement learning. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 3985–4003. Association for Computational Linguistics. https://doi.org/10.18653/v1/2020.emnlp-main.327

John F. Kelley. 1984. An iterative design methodology for user-friendly natural language office information applications. *ACM Transactions on Information Systems*, 2(1):26–41. https://doi.org/10.1145/357417.357420

W. Bradley Knox and Peter Stone. 2009. Interactively shaping agents via human reinforcement: the TAMER framework. In *Proceedings of the fifth international conference on Knowledge capture*, pages 9–16.

Alexander Koller, Kristina Striegnitz, Andrew Gargett, Donna Byron, Justine Cassell, Robert Dale, Johanna Moore, and Jon Oberlander. 2010. Report on the second NLG challenge on generating instructions in virtual environments (GIVE-2). In *Proceedings of International Natural Language Generation Conference*. Association for Computational Linguistics.

Robert M. Krauss and Sidney Weinheimer. 1966. Concurrent feedback, confirmation, and the encoding of referents in verbal communication. *Journal of Personality and Social Psychology*, 43:343–6. https://doi.org/10.1037/h0023705, PubMed: 5969163

Julia Kreutzer, Shahram Khadivi, Evgeny Matusov, and Stefan Riezler. 2018a. Can neural machine translation be improved with user feedback? In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 92–105. Association for Computational Linguistics. https://doi.org/10.18653/v1/N18-3012

Julia Kreutzer, Artem Sokolov, and Stefan Riezler. 2017. Bandit structured prediction for neural sequence-to-sequence learning. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1503–1513. Association for Computational Linguistics. https://doi.org/10.18653/v1/P17-1138

Julia Kreutzer, Joshua Uyheng, and Stefan Riezler. 2018b. Reliability and learnability of human bandit feedback for sequence-to-sequence reinforcement learning. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1777–1788. Association for Computational Linguistics. https://doi.org/10.18653/v1/P18-1165

Carolin Lawrence and Stefan Riezler. 2018. Improving a neural semantic parser by counterfactual learning from human bandit feedback. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 1820–1830. Association for Computational Linguistics. https://doi.org/10.18653/v1/P18-1169

Carolin Lawrence, Artem Sokolov, and Stefan Riezler. 2017. Counterfactual learning from bandit feedback under deterministic logging : A case study in statistical machine translation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 2566–2576. Association for Computational Linguistics. https://doi.org/10.18653/v1/D17-1272

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. Association for Computational Linguistics. https://doi.org/10.18653/v1/D17-1259

Bing Liu, Gokhan Tür, Dilek Hakkani-Tür, Pararth Shah, and Larry Heck. 2018. Dialogue learning with human teaching and feedback in end-to-end trainable task-oriented dialogue systems. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2060–2069. Association for Computational Linguistics. https://doi.org/10.18653/v1/N18-1187

Ilya Loshchilov and Frank Hutter. 2018. Decoupled weight decay regularization. In *Proceedings of the International Conference on Learning Representations*.

James MacGlashan, Mark K. Ho, Robert Tyler Loftin, Bei Peng, David L. Roberts, Matthew E. Taylor, and Michael L. Littman. 2017. Interactive learning from policy-dependent human feedback. In *Proceedings of the International Conference on Machine Learning*.

K. Mathewson and P. Pilarski. 2016. Simultaneous control and human feedback in the training of a robotic agent with actor-critic reinforcement learning. *arXiv*, abs/1606.06979.

Anjali Narayan-Chen, Prashant Jayannavar, and Julia Hockenmaier. 2019. Collaborative dialogue in Minecraft. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 5405–5415. Association for

Computational Linguistics. https://doi.org/10.18653/v1/P19-1537

Khanh Nguyen, Hal Daumé III, and Jordan Boyd-Graber. 2017. Reinforcement learning for bandit neural machine translation with simulated human feedback. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1464–1474. Association for Computational Linguistics. https://doi.org/10.18653/v1/D17-1153

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 311–318. Association for Computational Linguistics. https://doi.org/10.3115/1073083.1073135

P. M. Pilarski, M. R. Dawson, T. Degris, F. Fahimi, J. P. Carey, and R. S. Sutton. 2011. Online human training of a myoelectric prosthesis controller via actor-critic reinforcement learning. In *Proceedings of the International Conference on Rehabilitation Robotics*, pages 1–7. https://doi.org/10.1109/ICORR.2011.5975338, PubMed: 22275543

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. 2017. icarl: Incremental classifier and representation learning. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, pages 2001–2010. IEEE.

Anthony Robins. 1995. Catastrophic forgetting, rehearsal and pseudorehearsal. *Connection Science*, 7(2):123–146. https://doi.org/10.1080/09540099550039318

Yossi Rubner, Carlo Tomasi, and Leonidas J. Guibas. 1998. A metric for distributions with applications to image databases. In *Proceedings of the International Conference on Computer Vision*. IEEE.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv*, abs/1707.06347.

Burr Settles. 2009. Active learning literature survey.

Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F. Christiano. 2020. Learning to summarize with human feedback. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 3008–3021. Curran Associates, Inc.

Alane Suhr, Claudia Yan, Jack Schluger, Stanley Yu, Hadi Khader, Marwa Mouallem, Iris Zhang, and Yoav Artzi. 2019. Executing instructions in situated collaborative interactions. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 2119–2130. Association for Computational Linguistics. https://doi.org/10.18653/v1/D19-1218

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Proceedings of the Advances in Neural Information Processing Systems*, pages 3008–3021. Curran Associates, Inc.

Hao Tan, Licheng Yu, and Mohit Bansal. 2019. Learning to navigate unseen environments: Back translation with environmental dropout. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2610–2621. Association for Computational Linguistics. https://doi.org/10.18653/v1/N19-1268

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. Association for Computational Linguistics. https://doi.org/10.18653/v1/P16-1224

Yu-Xiang Wang, Alekh Agarwal, and Miroslav Dudík. 2017. Optimal and adaptive off-policy evaluation in contextual bandits. In *Proceedings of International Conference on Machine*

Learning, pages 3589–3597. Proceedings of Machine Learning Research.

Garrett Warnell, Nicholas R. Waytowich, Vernon Lawhern, and Peter Stone. 2018. Deep TAMER: Interactive agent shaping in high-dimensional state spaces. In *Proceedings of the AAAI Conference on Artificial Intelligence*.

Aaron Wilson, Alan Fern, and Prasad Tadepalli. 2012. A Bayesian approach for policy learning from trajectory preference queries. In *Proceedings of the Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Yang Xu and David Reitter. 2016. Convergence of syntactic complexity in conversation. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, pages 443–448. Association for Computational Linguistics

Dong Yu, Kaisheng Yao, Hang Su, Gang Li, and Frank Seide. 2013. Kl-divergence regularized deep neural network adaptation for improved large vocabulary speech recognition. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 7893–7897. IEEE.

Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. 2020. BERTScore: Evaluating text generation with BERT. In *Proceedings of the International Conference on Learning Representations*.

Ming Zhao, Peter Anderson, Vihan Jain, Su Wang, Alex Ku, Jason Baldridge, and Eugene Ie. 2021. On the evaluation of vision-and-language navigation instructions. In *Proceedings of the European Chapter of the Association for Computational Linguistics*, pages 1302–1316. Association for Computational Linguistics.

Zachary M. Ziegler, Luke Melas-Kyriazi, Sebastian Gehrmann, and Alexander M. Rush. 2019. Encoder-agnostic adaptation for conditional language generation. *arXiv*, abs/1908.06938.