A Limitations and Broader Impacts

Our study was conducted with a group of 108 workers, all recruited from English-majority locales, due to the complexity of recruiting and training workers given the complexity of the task. The group size limits the variation in language use we observe. Its composition restricts our ability to evaluate generalization to other languages, an important direction for future work. Another question for further study is the dynamics created when completely new users join the community in later stages.

Because of the complexity of our studies we kept our architecture close to previous work on CEREALBAR, and did not study more contemporary architectures or using pre-trained models. We hypothesize using both could lead to better performance, and more consistent improvement trends. This is an important direction for future work.

We do not vary the settings of CEREALBAR. Effenberger et al. [10] find that the interaction design and incentives influence the process of language change. Studying the impact of the scenario design decisions would have significantly complicate our experiments, and increase our costs. We decided not to focus on this research question in this work, but treat these parameters as fixed. Although the analysis of Effenberger et al. [10] shows that CEREALBAR creates interesting and complex language dynamics, further study of the impact of interaction design decisions on continual learning is important, and currently under-studied. Our work does not answer these questions, but we hope it will stimulate further research into them.

The data and models we release are not designed to be directly deployed beyond the CEREALBAR scenario. In general, deployment of continually learning systems requires guardrails and monitoring to avoid various undesired outcomes, including acquiring behaviors that may harm users.

B Model

We implement our policy as a neural network based on the design of Suhr et al. [35]. The inputs are an instruction $\overline{x}$ and observation o, and the output is a distribution over actions. The policy architecture is composed of several modules that combine to a single network.

Embedding Instructions We embed the instruction $\overline{x} = \langle x_1, \dots, x_n \rangle$ of length n with a bidirectional recurrent LSTM [13]. This results in a sequence of hidden states $\langle \mathbf{h}_1, \dots, \mathbf{h}_n \rangle$. The embedding of $\overline{x}$ is the final hidden state of the sequence $\mathbf{h}_n$.

Embedding Observations Each agent observation o includes information about the observable environment and the instruction execution so far. The follower agent in CEREALBAR has partial observability. We use a representation similar to that of Suhr et al. [35], but without making the simplifying assumption of full observability. The environment state $\mathbf{W}$ is a tensor representing the properties of each position in the environment as embedding indices. The properties represented in $\mathbf{W}$ also encode information about the follower's trajectory so far, the presence of obstacles in the environment, and the follower's observability. Due to partial observability, each position's representation is derived from its most recent observation; any information that changes about the world may be outdated in $\mathbf{W}$. We embed $\mathbf{W}$ into a dense tensor $\mathbf{W}'$.

Fusing Embeddings After independently embedding the instruction and observation into $\mathbf{h}_n$ and $\mathbf{W}'$, we compute a joint representation of both inputs using text-conditioned (i.e., via $\mathbf{h}_n$) convolutions over $\mathbf{W}'$.

Transforming the Coordinate System Predicting actions requires interactions between representations of multiple positions. $\mathbf{W}'$ represents the environment using offset coordinates, which do not precisely represent the structure of hexagonal grid in CEREALBAR. We transform $\mathbf{W}'$ to axial coordinates [14], and translate and rotate the tensor such that the center position represents the agent's current location, and the agent is facing in a consistent direction. These transformations are not parameterized.

LINGUNET We use LINGUNET [6] to predict the policy distribution over actions $\pi(\cdot \mid \overline{x}, o; \theta)$, with slight modifications to the design of Suhr et al. [35]. For all convolutions, we apply hex-based convolutions with kernels that operate only on voxels within a hex diameter of d around the center voxel, for a kernel size of d. We apply instance normalization to the last LINGUNET layer of the input and text-based convolutions. Finally, we do not perform the final transposed convolution. Instead, we directly predict a distribution over the action space given the output of the transposed convolution.

B.1 Inference

We use ensemble-based inference. Given sets of model parameters $\theta = \langle \theta_1, \ldots, \theta_m \rangle$, we construct a policy π over executable actions using voting:[11]

$$\pi(a \mid \overline{x}, o; \theta) \propto \exp\left(\sum_{1 \leq i \leq m} \mathbb{1}_{a = \arg\max \pi(\cdot \mid \overline{x}, o; \theta_i)} \right) . \tag{3}$$

Actions are sampled and executed from $\pi(\cdot \mid \overline{x}, o; \theta)$. Executing an action in the environment results in a observation according to the transition function $\mathcal{T}$. We continue to sample actions until the stop action `STOP` is sampled, or until the leader manually reboots the follower. The `STOP` action marks the current instruction as complete, which either results in the follower's turn ending, or it receiving the next instruction to follow.

C Implementation Details

We lowercase and tokenize instructions using BPE [31] with a maximum vocabulary size of 4,096 and a minimum wordtype occurrence of 2.[12] We learn size-64 word embeddings from scratch. We encode instructions with a single-layer LSTM RNN [13] with 128 hidden units. We embed each position's properties into vectors of size 16. We use the same LINGUNET hyperparameters as Suhr et al. [35], and did not perform an additional hyperparameter search.

We use an ensemble size of $m = 10$. We do not train in ensemble, but train ten separate models and apply ensemble-based inference during deployment. When using reward propagation, we use a maximum distance of 8 for propagating to previous actions that received no feedback. For training, we use a batch size of 16 agent steps, a learning rate of 0.001, and ADAM [17] for optimization. We re-initialize model parameters from scratch at the beginning of each round of parameter optimization. We use a held-out subset of the original CEREALBAR training set as a validation set for early stopping, comprising 5% of the original split. After each epoch, we evaluate model performance using SWSD (Appendix D) on the validation set. We use patience for stopping; if ten epochs have passed since the last epoch where the validation SWSD surpassed the previous global maximum, we terminate the training process and choose the model parameters that maximize validation SWSD. We use a single GeForce RTX 2080 Ti for training each model. Training a single model takes about 28.9 hours on average. We run inference on CPU during deployment.

Comparison of Learning Design Choices In our second experiment comparing different learning design choices, we deploy for five rounds. This number of rounds was chosen because after five rounds in the long-term experiment (Section 5.1), learning trends were clear; this choice balances experiment cost and insight. If we acquire more than 200 interactions per round because of the crowdsourcing process, we select exactly 200 games for training and analysis by preferring earlier games played by each worker. We discard the other games.

D Evaluation

Instruction Execution Accuracy For each deployed agent, we randomly sample instruction execution traces $\mathcal{E}_e \subseteq \mathcal{E}_c \subseteq \mathcal{E}$ for manual evaluation. $\mathcal{E}_c$ contains all instructions marked as complete by the agent, and $\mathcal{E}$ contains instructions that were either marked as complete or rebooted.[13] Excluding rebooted instructions from this evaluation creates a biased sample, as reboots nearly always reflect

[11] We assign zero probability to inexecutable actions, i.e., one that would result in an intersection with an obstacle.

[12] We use the implementation provided by HuggingFace at https://huggingface.co/docs/tokenizers/.

[13] In this evaluation, we ignore all instructions that were not completed due to the game ending.

incorrect instruction execution, so we re-adjust accuracy estimates based on reboot rates. We assume all rebooted instructions are incorrect executions. The adjusted correctness rate is:

$$\text{correctness} = \frac{\sum_{\overline{e} \in \mathcal{E}_e} \mathbb{1}_{\text{correct}(\overline{x}, \overline{e})}}{\mid \mathcal{E}_e \mid} \frac{\mid \mathcal{E}_c \mid}{\mid \mathcal{E} \mid} ,$$

where correct$(\overline{x}, \overline{e})$ is user judgment of execution $\overline{e} = \langle (o_i, a_i, w_i^a) \rangle_{i=1}^m$ for instruction $\overline{x}$.

Static Evaluation Data We also evaluate on the development split from Suhr et al. [35], with *success weighted by stopping distance* (SWSD). SWSD is computed per instruction execution:

$$\text{SWSD}(\overline{e}', \overline{e}^*) = \frac{\mathbb{1}_{\overline{e}'_{-1} \equiv \overline{e}^*_{-1}}}{1 + \mid\mid \overline{e}'_{-1} - \overline{e}^*_{-1} \mid\mid} ,$$

where $\overline{e}'$ is the trace of the agent's execution of an instruction and $\overline{e}^*$ is the human demonstration. $\overline{e}'_{-1} \equiv \overline{e}^*_{-1}$ only if $\overline{e}'$ results in the same set of cards selected as in $\overline{e}^*$. $\mid\mid \overline{e}'_{-1} - \overline{e}^*_{-1} \mid\mid$ is the hex distance between stopping positions. SWSD is stricter than simple card-state accuracy [35], as it gives only partial credit to instructions where an execution stops in an incorrect position.

E Crowdsourcing and Data Details

This study received an exemption from the Institutional Review Board of the institution where the research was conducted. Worker identities are anonymized in the data we release.

We qualify workers through a tutorial and a short quiz about the game rules. Workers are also required to reside in an English-majority locale and have a HIT approval rate of over 90% with at least 100 approved HITs during their time on MTurk. The base pay for completing the qualification task is \$0.50 USD, and qualified workers receive a \$2.00 bonus. We qualify 108 workers. Following Suhr et al. [35], we pay workers bonuses per point earned in each game, increasing the compensation per point as the game score increases. On average across all experiments, each game costs \$2.91 and workers are paid an average of \$21.85 per hour.

We split workers into two pools: expert and novice. Expert workers earn a 50% higher bonus per game than novice workers. Workers are moved to the expert pool after playing at least two games with a score greater than zero, as long as their rate of giving feedback is greater than 75% of instructions.[14] Workers return to the novice pool if they play for two rounds with a feedback rate of less than 75% of instructions. 65 workers achieve and maintain expert status throughout the experiments. Only expert workers are qualified to provide post-hoc instruction execution judgments, where they are paid \$0.07 per judgment of instruction execution. Worker IDs are anonymized in the distribution of interaction data. Figure 5 shows the instructions provided to workers in MTurk, and Figure 6 shows the CEREALBAR interface during interaction.

Agreement For about 20% of instruction execution receiving post-hoc judgments, we acquire judgments from three workers; we find that for only 3.6% of these no consensus is achieved among the three workers, which indicates very high overall agreement.

F Additional Results

F.1 User Perception of Agents for Comparison of Learning Design Choices

Figure 7 shows the Likert distribution for the three post-interaction statements users are asked about for the experiments comparing learning design choices, where we concurrently deployed five systems for five rounds.

F.2 Evaluation on Static Data

Evaluation through human-agent interaction is the main focus of our work. However, we also evaluate instruction-following agents against held-out, static data from Suhr et al. [35]. This evaluation does

[14] The rate of feedback per instruction measures the proportion of instructions where at least one action in the follower's instruction execution is given positive or negative feedback, including a reboot.

In this HIT, you'll play a collaborative game with a partner.

- You will be a *Leader* and will work with an automated *Follower* (i.e., not another worker).
- You should be able to connect quickly to your partner. If you spend more than a minute or two waiting for a partner, please contact us.
- If you are confused about the game, please read the instructions on the game page, or visit this page. You can also email us at [REDACTED FOR SUBMISSION].

Check the HIT title to see whether you are in the *novice* or *expert* group.

HITs in the expert pool have about a 50% increase in bonus amount over to the novice pool. See the instructions page for details and the bonus scheme for novice and expert workers.

We will evaluate workers in the expert batch periodically to make sure they are still adhering to our expert standards, and we will evaluate novice workers after every batch to see if they can move into the expert pool.

- Some things we will take into account for the Leader games:
  - You should move around the map with your partner to collect sets.
  - Your commands should be unambiguous and easy for your partner to follow.
  - You should understand what makes a valid set (three cards with different colors, shapes, and numbers).
  - You should be actively giving good/bad feedback to your partner (at least a few signals per instruction).
  - You shouldn't use the Reboot button unless the Follower is performing poorly.
  - **Give feedback for over 75% of instructions (feedback can include rebooting the follower).**
- To get into the expert pool, novices need to complete at least 2 games with a score of at least 1, and give feedback for at least 75% of instructions.

Please report any issues you see in the feedback box below, and take screenshots where possible (email to [REDACTED FOR SUBMISSION]).

Don't attempt to play more than one game at a time. Our server will only accept a single connection from each worker at a time. We appreciate your full attention while playing the game!

Figure 5: Instructions provided to workers for the main task on MTurk. Detailed instructions about gameplay were provided on a separate set of webpages, and will be available alongside our code when released.

Figure 6: The CEREALBAR interaction interface. Users provide instructions in the command box, and feedback via either buttons in the GUI or keypresses. The follower's partial view of the environment is visible in the bottom righthand corner of the interface.

not take into account how the actual data distribution shifts over the agent's lifetime, because of the dynamics between the agent and human users. Figure 8 shows average SWSD for the models deployed in each round. SWSD begins at 39.7 for the initial model, and peaks at 46.4. This improvement is due entirely to adding training data acquired from human-agent interactions.

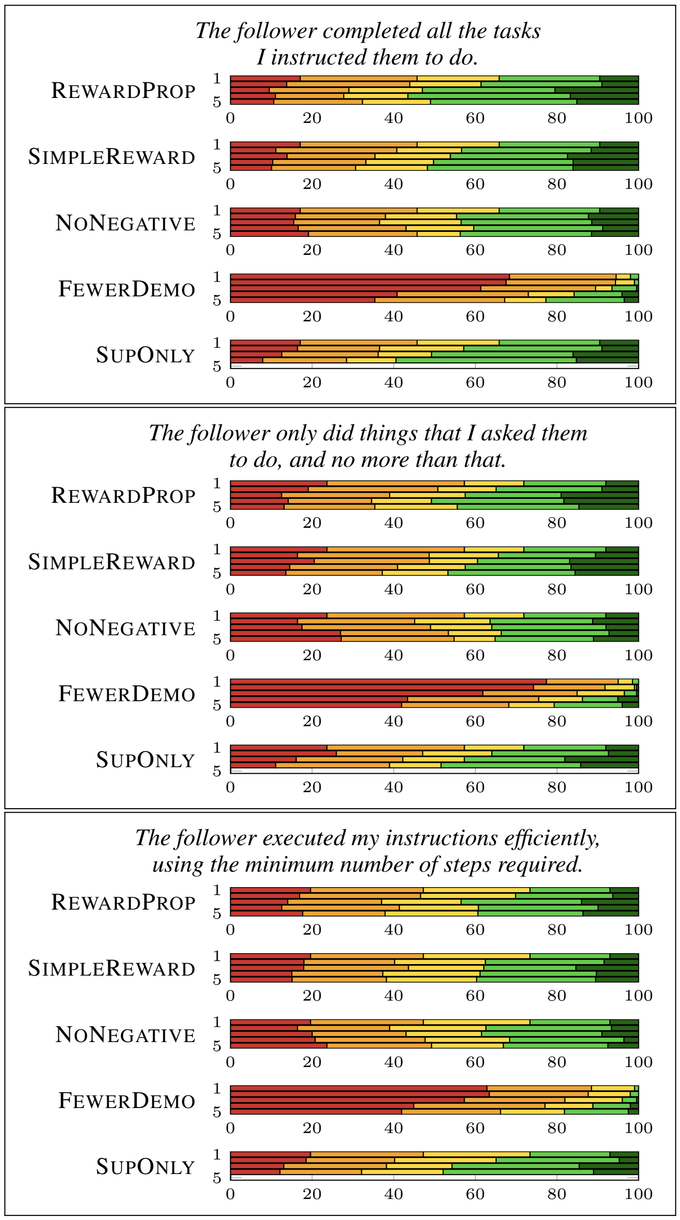

Figure 7: Distribution of post-interaction user agreement with three statements about the follower's performance for our approach comparison experiment.

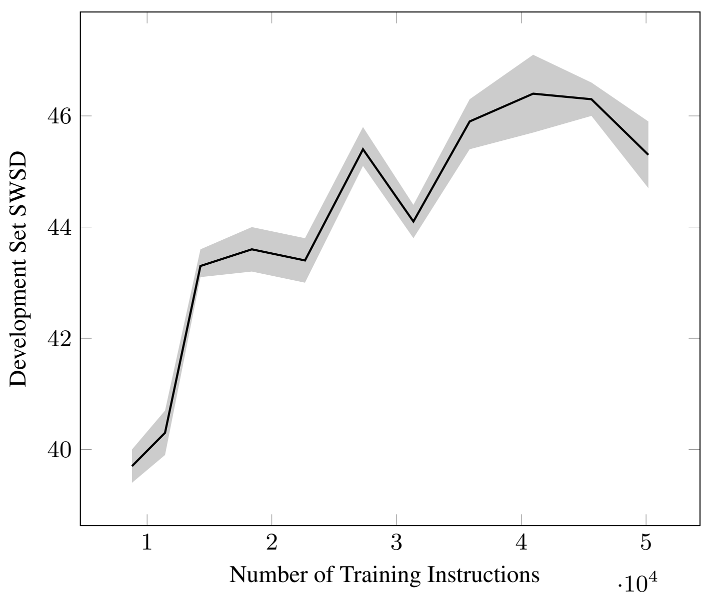

Figure 8: SWSD on the held-out development data, averaged over five runs of sampling-based inference.