Learning to execute instructions in a Minecraft dialogue

Prashant Jayannavar **Anjali Narayan-Chen** **Julia Hockenmaier**
University of Illinois at Urbana-Champaign
{paj3, nrynchn2, juliahmr}@illinois.edu

Abstract

The Minecraft Collaborative Building Task is a two-player game in which an Architect **A** instructs a Builder **B** to construct a target structure out of 3D blocks. We consider the task of predicting **B**'s action sequences (block placements and removals) in a given game context, and show that capturing **B**'s past actions as well as **B**'s perspective leads to a significant improvement in performance on this challenging language understanding problem.

1 Introduction

There is a long-standing interest in building interactive agents that can communicate with humans about and operate within the physical world (e.g. Winograd (1971)). The goal for agents in this scenario is to not only be able to engage in rich natural language discourse with their human conversation partners, but also to ground that discourse to physical objects, and execute instructions in the real world. Traditional dialogue scenarios are either completely ungrounded (Ritter et al., 2010; Schrading et al., 2015), focus on slot-value filling tasks (Kim et al., 2016b,a; Budzianowski et al., 2018) which instead require grounding to entities in a knowledge base, or operate within static environments, such as images (Das et al., 2017) or videos (Pasunuru and Bansal, 2018). Relevant efforts in robotics have largely focused on single-shot instruction following, and are mostly constrained to simple language (Roy and Reiter, 2005; Tellex et al., 2011) with limited resources (Thomason et al., 2015; Misra et al., 2016; Chai et al., 2018).

The recently introduced Minecraft Collaborative Building Task and the corresponding Minecraft Dialogue Corpus (Narayan-Chen et al., 2019) is one attempt to bridge this gap within the simulated game world of Minecraft. In this task, two players, an Architect (**A**) instructs a Builder (**B**) to construct a target structure out of multi-colored building blocks. The corpus consists of 509 game logs between humans that perform this task. Narayan-Chen et al. (2019) focus on generating Architect utterances. In this paper, we explore models for building an automated Builder agent.[1] We focus on the subtask of predicting the Builder's block placements, and leave the back-and-forth dialogue aspect of the overall task required of a fully interactive Builder agent to future work. We define the Builder Action Prediction (BAP) task in Section 2, describe our models in Section 3, an approach to augment the training data in Section 4, and our experiments in Section 5. We analyze results and highlight challenges of the BAP task in Section 6.

2 Dataset and Task

2.1 The Minecraft Dialogue Corpus

The Minecraft Dialogue Corpus (Narayan-Chen et al., 2019) consists of 509 human-human dialogues and game logs for the Minecraft Collaborative Building Task, a two-player game in a simulated Blocks World environment between an Architect (**A**) and a Builder (**B**). **A** is given a target structure (*Target*) and has to instruct **B** via a text chat interface to build a copy of *Target* on a given build region. **A** and **B** communicate back and forth via chat throughout the game (e.g. to resolve confusions or to correct **B**'s mistakes), but only **B** can move blocks, while **A** observes **B** operating in the world. **B** is given access to an inventory of 120 blocks of six given colors that it can place and remove. The resulting dialogues consist mainly of **A** providing instructions, often involving multiple actions to be taken, and grounded in the Builder's perspective, while **B** executes those instructions and resolves

[1] For models and code see http://juliahmr.cs.illinois.edu/Minecraft

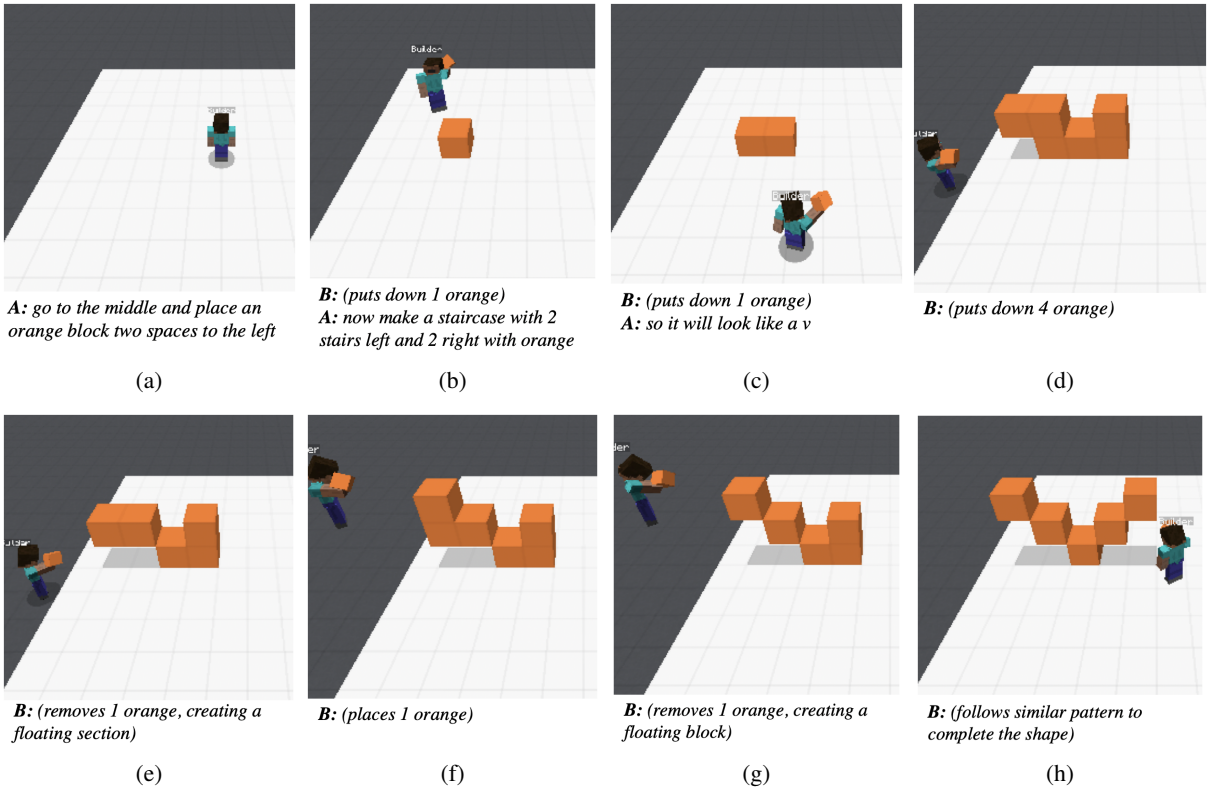

Figure 1: A sample sequence of human-human game states. The game starts with an empty grid and an initial **A** instruction (a), which **B** executes in the first action sequence (b) by placing a single block. In (c), **B** begins to execute the next **A** instruction given in (b). However, **A** interrupts **B** in (c), leading to two distinct **B** action sequences: (b)–(c) (single block placement), and (c)–(h) (multiple placements and removals).

any confusion through further dialogue. The task is complete when the structure built by **B** (*Built*) matches *Target* (allowing for translations within the horizontal plane and rotations about the vertical axis) and lies completely within the boundaries of the predefined build region. Games in this corpus are based on 150 distinct target structures, split into disjoint test, training, and development sets such that training targets do not appear during test or development. Game logs record all utterances and **B**'s actions (placements and removals), as well as the state of the world (i.e. the (x,y,z)-coordinates and colors of all blocks in the build region), and **B**'s (x,y,z) position, vertical rotation (pitch) and horizontal orientation (yaw) at the points in time when an utterance was recorded or an action performed. Since there are six block colors to be placed, we distinguish seven possible types of actions $A \in \{\text{BLUE}, \text{GREEN}, ..., \text{YELLOW}, \text{REMOVE}\}$. **B** actions are 4-tuples $\langle A, x, y, z \rangle$ consisting of an action type and cell coordinates. A block placement is feasible as long as an adjacent grid location is occupied, while REMOVE is feasible as long as that location is currently occupied by a block. These actions do not include **B**'s movement. **B** can assume any (continuous) 3D position and orientation, and the dataset records **B**'s position and orientation for each individual action. But since there are many positions and orientations from which blocks in a cell can be placed, **B**'s movement is secondary to the main task of constructing the target configuration.

2.2 The Builder Action Prediction Task

Narayan-Chen et al. (2019) focused on creating models that can generate **A** utterances, whereas we aim to develop models that can perform **B**'s role. Although back-and-forth dialogue between the two players is a clear hallmark of this task, we leave the question of how to develop **B** agents that can decide when to speak and what to contribute to the conversation (either by way of chit-chat, verifications or clarification questions to **A**) to future work, and focus here on the subtask of predicting correct sequences of block placements and removals. Executing **A** instructions is **B**'s primary role, and a crucial component to overall task completion.

Figure 1 shows an example from the Minecraft Dialogue Corpus that highlights some challenges

of performing this task. **A** can move around freely, but remains invisible to **B** and views the structure from behind **B** when giving instructions. As a result, **A** instructions frequently include spatial relations, both between pairs of blocks or substructures (*"put ... on top of.."*), and relative to **B**'s current position and perspective (*"left"*, *"right"*). **A** also often uses higher-level descriptions involving complex shapes (e.g. *"staircase"*, *"v"*). Due to the asynchronous nature of the dialogue, **A** often interrupts during **B** action sequences. **A** may also provide corrections and clarifications to fix **B** mistakes. Producing feasible sequences of **B** actions requires a certain amount of planning, since blocks can only be placed in grid cells that are adjacent to other blocks or the ground, and floating structures (a common occurrence among the target structures in this corpus) can only be built if supporting blocks that are not part of the target structure are present when the floating blocks are being placed. Despite these challenges, we show below that training models that use a rich representation of the world (Section 3) on sufficient amounts of diversified data (Section 4) produces promising initial results.

To generate items for this task, we follow a similar strategy to Narayan-Chen et al. (2019), who, as a first step towards designing a fully interactive Architect, define an Architect Utterance Generation Task, where models are presented with a particular human-human game context in which a human Architect produced an utterance and are evaluated based on how well they can generate an appropriate utterance. Conversely, we define the Builder Action Prediction (BAP) Task as the task of predicting the sequence of actions (block placements and/or removals) that a human Builder performed at a particular point in a human-human game.

2.3 Evaluating Builder Action Predictions

To evaluate models for the BAP task, we compare each model's predicted action sequence A_m against the corresponding action sequence A_h that the human builder performed at that point in the game. Specifically, for each pair of model and human action sequences (A_m, A_h), where $A_h = \langle a_h^{(1)}, ...a_h^{(k)} \rangle$ led from a world state W_{before} to a world state W_h and $A_m = \langle a_m^{(1)}, ...a_m^{(l)} \rangle$ led from the same W_{before} to W_m, we compute an F1 score over the net actions in A_h and A_m, and report a micro-average over all sequences in the test (or development) data.

Net actions ignore actions that were undone within the same sequence, e.g. if a block was placed and then removed. We consider any a_m action correct if the same action (involving the same grid cell and block color) occurs among the net actions in A_h. There are two reasons why we evaluate net rather than all actions: first, many structures contain floating blocks which require the placement of temporary "placeholder" blocks that are later removed. Placeholders' colors are arbitrary, and there are often multiple possible locations where placeholders can be put; placeholder predictions should not be penalized, as long as they enable the correct target to be built. Human Builders are also prone to making small mistakes that are immediately resolved (e.g. by removing blocks that were accidentally placed). Evaluation should be robust to this noise in the ground truth sequences.

The F1 metric ignores sequence information because it is either implicit in cases where it matters (e.g. building a vertical stack of blocks from the ground up), or irrelevant (e.g. building a line of blocks on the ground). Other metrics may also be suited for this task, but obvious choices such as an edit distance between W_m and W_h suffer from the problem that they favor models that place fewer blocks, since incorrect placements would incur twice the cost of no placements. However, our current definition of when an action is correct is relatively harsh, and could be relaxed in a number of ways. First, since it only considers an action correct if it matches a human action at the same grid cell, it penalizes cases where there are rotational equivalences between the built and the target structures (as may arise when the target has rotational symmetry). It also ignores any translational equivalences (which are very common at the beginning of a dialogue when the initial structure is empty, and may also need to be taken into account when the action sequence passes through an intermediate state in which all blocks have been removed). Second, looser F1 scores that evaluate actions only with regard to block locations (ignoring color) or colors (ignoring locations) might yield insight into how well models understand spatial relations, colors, or the number of blocks to be placed or removed. We leave exploring such variants to future work.

While our evaluation allows us compare models directly and automatically against a common gold standard, it is important to keep in mind that such direct comparisons to human action sequences pro-

vide only a lower bound on performance because they are based on the assumption that a) the human executed the instructions completely and correctly, and that b) there is only one way to execute the instructions correctly. But instructions are often vague or ambiguous: *"Place a red block on the ground next to the blue block"* may be resolved to any of four equally correct cells adjoining that block, and ideally, the evaluation metric should score them the same. And human action sequences do not always correspond to a complete execution of the previous instruction, e.g. when **B** is interrupted by **A** or stops to ask a question:

A: now it will be a diagonal staircase with 4 steps angling towards the middle
A: if that makes sense
B *puts down a red block*
B: diagonal staircase with this orientation?
B *puts down a red block*
A: towards where the yellow blocks are pointing
B *picks up 2 red blocks, puts down a red block*

2.4 Related Work

There is growing interest in situated collaborative scenarios involving instruction givers/followers with one-way (Hu et al., 2019; Suhr et al., 2019) and two-way (Kim et al., 2019; Ilinykh et al., 2019) communication. Here, we compare our task to related work on instruction following, both generally and within Blocks World and Minecraft.

Instruction following: Prior approaches to instruction comprehension typically take a semantic parsing approach (Chen and Mooney, 2011; Artzi and Zettlemoyer, 2013; Andreas and Klein, 2015). Semantic parsing components enable human-robot understanding (Tellex et al., 2011; Matuszek et al., 2013); some approaches to interactive robot design combine these architectures with physical robot exploration to enable online learning (Thomason et al., 2015, 2016, 2017). The SCONE corpus (Long et al., 2016) features tasks in three domains requiring context-dependent sequential instruction understanding, in which a system is given a world containing several predefined objects and properties and has to predict the final world state by parsing instructions to intermediate logical forms. Some papers have also applied neural action prediction models (Suhr and Artzi, 2018; Huang et al., 2019) to SCONE. More recently, Vision-and-Language Navigation (VLN), (Anderson et al., 2018), and its dialog counterpart, Cooperative Vision-and-Dialog Navigation (CVDN) (Thomason et al., 2019), focus on instruction following and cooperative interactions in photorealistic navigation settings.

Since our dataset does not contain any logical forms, we also cannot use semantic parsing approaches, and have to resort to neural action prediction models. However, Minecraft instructions are more challenging than the SCONE tasks because our action space is significantly larger and our utterances are more complex. Minecraft dialogues are also more complex than the sequences of instructions in SCONE because we cannot assume that actions to be executed are described in the last utterance. Minecraft dialogues are also more complex than those in CVDN, because they contain more turns, and because communication is asynchronous. Moreover, construction differs fundamentally from navigation in that construction dynamically changes the environment. While referring expressions in navigation can be safely assumed to refer to objects that exist in the world, construction instructions frequently refer to objects that need to be built by the agent. And although more recent navigation tasks require real vision, their underlying world state space (as defined by fixed viewpoints and the underlying navigation graph) is just as highly discretized. Our task does not require vision, but poses an arguably more challenging planning problem, since its action space is much larger (7623 possible actions vs. six actions in the vision-language navigation work).

Blocks World: There is a renewed interest in instruction comprehension in Blocks World scenarios. Voxelurn (Wang et al., 2017) interfaces with human users and learns to understand descriptions of voxel structures of increasing complexity, but does so by mapping them down to a core programmatic language. Bisk et al. (2016a,b, 2018) build models for understanding single-shot instructions that transform one world state to another using simulated 3D blocks. Blocks are viewed from a fixed bird's-eye perspective, initialized randomly in the initial world state, and uniquely identifiable. The varying Builder perspective and lack of easily identifiable referents, along with the need to understand utterances in a dialogue context, make our task a much more challenging problem. Unlike traditional Blocks World, Minecraft allows blocks to float (requiring nonmonotonic action sequences where placement is followed by removal), or attach to any side of an existing block.

Minecraft: Combining semantic parsing with simulated human-robot interaction, Facebook CraftAssist is a dialogue-enabled framework with an associated dataset for semantic parsing of instructions in Minecraft (Gray et al., 2019; Jernite et al., 2019; Szlam et al., 2019). Their setup enables two-way human-bot interactions in which a human architect can direct an automated builder using natural language to build complex structures. To bootstrap a semantic parser, they synthetically generate (using a hand-defined grammar) and crowdsource natural language instructions paired with logical tree structures consisting of action primitives. In addition to lacking such annotations, our work differs fundamentally in that our data is sourced from human-human dialogues; instructions are more ambiguous, dialogues have larger variety and Builder action sequences are noisier.

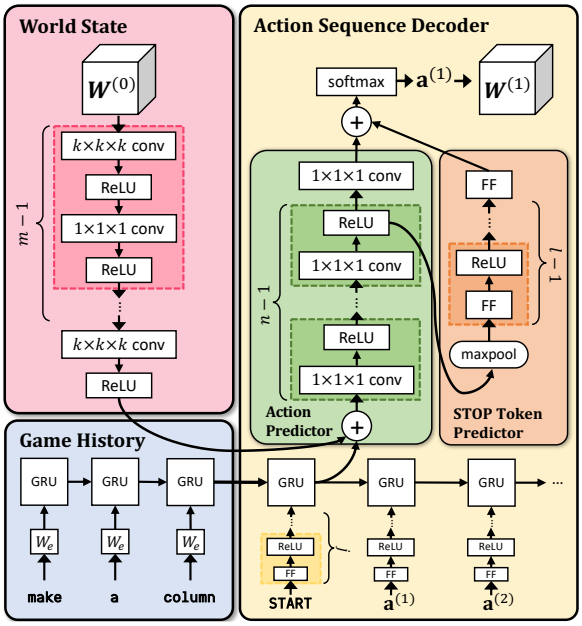

Figure 2: The Builder Action Prediction model.

3 Builder Action Prediction Models

3.1 Overall architecture

Similar to e.g. the models of Suhr and Artzi (2018) for the SCONE tasks, models for the Builder Action Prediction task need to predict an appropriate, variable-length, sequence of actions (block placements and removals) in a given discourse and game context and world state. All our models (Figure 2) are based on a recurrent encoder-decoder architecture (Sutskever et al., 2014; Cho et al., 2014) in which a GRU-based encoder (bottom left box) captures the game context (dialogue and action history), and a CNN-based encoder (top left box) captures the world state at each time step. The decoder (right box) predicts one action per time step, based on the game history, the world state at that time, and the last action taken. It consists of another GRU backbone over action sequences (bottom right), and a multi-class classifier that reads in the output of the GRU backbone as well as the world state encoding produced by the CNN to predict either the next action (block placement or removal) to be taken, or a special STOP token that terminates the action sequence. The world state representation gets updated and re-encoded after each predicted action. We now describe these components in more detail.

3.2 Game history encoder

Since **B** only knows what blocks to place after receiving an instruction from **A**, we can view the game history as a non-empty sequence of previous utterances (by both players), possibly interleaved with sequences of actions that were taken by **B** in earlier turns of the game. Our experiments examine the question of how much of this history should be given to our model, but all models examined in this paper treat the game history as a single sequence of tokens. Similar to Narayan-Chen et al. (2019), we encode the dialogue history as a sequence of tokens in which each player's utterances are contained within speaker-specific start and end tokens ($\langle A\rangle \ldots \langle\backslash A\rangle$ or $\langle B\rangle \ldots \langle\backslash B\rangle$.). We also represent **B**'s prior actions naively as tokens that capture the action type (placement or removal) and block color (e.g. as "builder_putdown_red"). The $2 \times 6 = 12$ action tokens as well as the speaker tokens are encoded using 300-dimensional random vectors, while all other tokens are encoded as 300-dimensional pre-trained GloVe word embeddings (Pennington et al., 2014). The token embeddings are passed through a GRU to produce a H-dim embedding ($H \in \{200, 300\}$) of the dialogue history in the GRU's final hidden state.

3.3 World state encoder

The world state is the current grid configuration that is fed into the action prediction model at each time step. We first describe how we represent the raw world state, before we explain how this representation is then encoded via a CNN-based architecture.

Input: the raw world state Minecraft blocks are unit cubes that can be placed at integer-valued

$\langle x, y, z \rangle$ locations in a 3D grid; the Collaborative Building Task restricts these to a build region of size $11 \times 9 \times 11$. Since we found it beneficial to explicitly capture empty grid cells, our baseline model represents each cell state as a 7-dim one-hot vector, yielding a $11{\times}9{\times}11{\times}7$ minimal world state representation encoding the presence (or absence) of blocks at any grid cell. We also found it useful to capture the relative position of each cell with respect to **B**'s current position and orientation, as well as which cells were affected by **B**'s most recent actions, and augment this model in two ways:

Action history weights: Each action affects a single grid cell. Actions that follow each other often affect adjacent grid cells. We encode information about the most recent actions in our world state representation as follows: Given the chronological sequence of all actions $A = a^{(1)}, a^{(2)}...a^{(t-1)}$ that took place before the t-th action to be predicted, we assign a real-valued weight $\alpha^{(i)}$ to each action $a^{(i)}$ (where $\alpha^{(i)} \leq \alpha^{(i+1)}$), and include these action weights in the world state representation of the corresponding cells. We truncate the action history to the last five elements, assign integer weights 1...5 to $a^{(t-5)}, ..., a^{(t-1)}$ (and 0 to all $a^{(i<t-5)}$), and then include these weights as a separate input feature in each cell. If a cell was affected more than once by the last five actions, we only use the weight of the most recent action. Our action weights do not distinguish between actions taken in the preceding action sequence and those in the current sequence.

Perspective coordinates: **B** needs to understand the spatial relations in **A**'s instructions. Many of these relations (e.g. *"left"* in Figure 1) depend on **B**'s current position $\langle x_B, y_B, z_B \rangle$ and orientation (pitch $\phi_B \in [-90, ..., +90]$, or vertical rotation, and yaw $\gamma_B \in [-180, ..., +180]$, horizontal orientation). Our models assume that spatial relations in an instruction are relative to **B**'s position at that time, and use that information to compute perspective coordinates. We calculate the relative perspective coordinates $\langle x'_c, y'_c, z'_c \rangle$ of a cell c with absolute coordinates $\langle x_c, y_c, z_c \rangle$ by moving the frame of reference from $\langle 0, 0, 0 \rangle$ to $\langle x_B, y_B, z_B \rangle$, and rotating it to account for **B**'s yaw and pitch:[2]

$$\langle x'_c, y'_c, z'_c \rangle = P \cdot Y \cdot \langle x_c - x_B, y_c - y_B, z_c - z_B \rangle$$

We scale these perspective coordinates by a factor of .1 to keep their range closer to that of the cell state and action history weights.

Our full model represents each cell as an 11-dim vector (consisting of the 7-dim cell state, 1-dim action history weight and 3-dim perspective coordinates), and the entire grid (which serves as input to a CNN-based encoder) as a $11{\times}11{\times}9{\times}11$ tensor. We refer to the grid at time step t as $W^{(t)}_{raw}$.

Output: a CNN-based encoding To obtain a representation of each grid cell, we feed the raw world state tensor $W^{(t)}_{raw}$ of Section 3.3 through a multi-layer CNN that embeds each grid cell conditioned on its neighborhood and recent actions (if using action history weights). The model consists of m 3d-conv layers with kernel size 3 (CNN_3), stride 1 and padding 1, followed by a ReLU activation function. Between every successive pair of these layers is a $1 \times 1 \times 1$ 3d-conv layer (CNN_1) with stride 1 and no padding, for dimensionality reduction purposes, again followed by ReLU. With $W^{(t)}_0 = W^{(t)}_{raw}$, the first $m - 1$ blocks of this model can be expressed as $W^{(t)}_i = \text{relu}(\text{CNN}^i_1(\text{relu}(\text{CNN}^i_3(W^{(t)}_{i-1}))))$. The m'th 3×3×3 3d-conv layer CNN^m_3 computes the final world state representation $W^{(t)}_m = \text{relu}(\text{CNN}^m_3(W^{(t)}_{m-1}))$ that is used to predict the next action.

3.4 Action Sequence Decoder

The GRU backbone The GRU backbone of the decoder captures information about the current action sequence and the game history. We initialize its hidden state with the final hidden state of the game history encoder RNN of Section 3.2. Since the tensor representation of the grid is too unwieldy to be used as input to a recurrent net, we instead compute an explicit 11-dim representation $\mathbf{a}^{(t-1)}$ of the action taken at the last time step, consisting of three components: a 2-dim one-hot vector for the **action type** (placement or removal), a 6-dim one-hot vector for the **block color** (all zero for removals), and a 3-dim **block location** vector containing the absolute $\langle x, y, z \rangle$ coordinates of the cell where the action took place. At the start of decoding, we use a zero vector as a start token. These action vectors get passed through j dense linear layers with ReLU before being fed to the GRU.

Output: Next action prediction With seven possible actions per cell, there are 7623 possible actions (although only a small subset of these will be feasible at any point in time, a point that we will return to below). Since our models need to

[2] $P = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi_B & \sin\phi_B \\ 0 & -\sin\phi_B & \cos\phi_B \end{pmatrix}$ and $Y = \begin{pmatrix} \cos\gamma_B & 0 & -\sin\gamma_B \\ 0 & 1 & 0 \\ \sin\gamma_B & 0 & \cos\gamma_B \end{pmatrix}$

predict a variable length sequence of actions, we also need a special STOP action that is not associated with a single cell, but terminates the sequence. Our action prediction classifier has therefore two sub-components: a block action prediction model, and a stop prediction model. The stop prediction model returns a single element, which we append to the vector returned by the block action prediction model before feeding it through a softmax layer to return the most likely next action.

Block actions scores: We use a CNN-based architecture with parameter sharing across cells to score each of the seven possible actions for every grid cell. The input to this model consists of the CNN-based world state representation $W_m^{(t)}$ (Section 3.3), as well as the decoder GRU's hidden state $\mathbf{h}^{(t)}$, concatenated to each cell's representation in $W_m^{(t)}$ as additional channels. This model consists of $n-1$ 1×1×1 3d-conv layers followed by ReLU ($W_i'^{(t)} = \text{relu}(\text{CNN}_1^i(W_{i-1}'^{(t)})$) and with the nth such 3d-conv layer with 7 output channels (and no ReLU): $W_n'^{(t)} = \text{relu}(\text{CNN}_1^n(W_{n-1}'^{(t)}))$, which is flattened into a 7623-dim vector of action scores.

STOP score: We also need to predict when an action sequence is complete. While this decision needs access to the same information as the block action scorer, it also needs access to a (compact) global representation of the grid, since the STOP action is not cell-specific. It also needs to know the uncertainty in the block action scorer, since STOP is more likely when it is less clear which block action should be performed, and vice versa. We take the output of the penultimate layer in the block action scorer and apply max-pooling to every cell's vector representation, thus obtaining a single number for each of the 1089 cells. We concatenate these numbers into a single vector and use that as input to the STOP prediction model, which consists of l dense linear layers (with ReLU after each layer except the last), where the lth layer has a single output $W_l''^{(t)}$, the score for STOP.

Final action prediction scores: Finally, we concatenate the block action and STOP scores and apply a softmax to obtain the final prediction $a^{(t)}$:

$$a_t = \arg\max(\text{softmax}(vec(W_n'^{(t)}) \oplus W_l''^{(t)}))$$

4 Data Augmentation

The small size of the training set (3,709 examples) is a major limiting factor for training complex models. Here, we explore ways of generating synthetic data to augment the size and variety of our data. For each game log in the original training data, we generate twenty new game logs by combining the following data augmentation techniques:

Utterance paraphrases: We generate paraphrases of the utterances in the dialogue by randomly substituting tokens with any of their synonyms in the hand-engineered synonym lexicon of Narayan-Chen et al. (2019).

Color substitutions: We permute block colors by applying one of the 6! possible permutations, chosen at random, to the entire game log. These substitutions also change the language in the synthetic dialogues to reflect the updated colors.

Spatial transformations: Since the world contains no landmarks besides the built region, absolute coordinates are somewhat arbitrary. We sample one (0, 90, -90, 180) rotation in the ground plane (affecting all $\langle x, z\rangle$ coordinates, plus **B**'s yaw and position) per synthetic log (subject to the constraint that the target still fit in the built region).

5 Experiments

We evaluate our world state encoders, game history and data augmentation schemes.

Experimental Setup Our training, test and development splits contain 3709, 1616, and 1331 Builder action sequences respectively. We increase the training data to 7418 (2x), 14836 (4x) and 22254 (6x) items by sampling items from the synthetic data of Section 4. The average sequence length (in the development set) is 4.3 (with a std. deviation of 4.5). Target structures in the test data do not appear in the training or development data. We train models with AdamW (Loshchilov and Hutter, 2019) and weight decay regularization with a weight decay factor of 0.1. We use a learning rate of 0.001 for the original data and a slightly lower learning rate of 0.0001 in the case of augmented data. We use a batch size of 1. During training, we use teacher forcing and minimize the sum of the cross entropy losses between each predicted and ground truth action sequence (the action sequence performed by the human). We stop training early when loss on the held-out development set has increased monotonically for ten epochs. We use greedy decoding (max. sequence length of 10) to generate action sequences, which seems to work better than beam search decoding (for fixed beam sizes between 5 and 20). We report net action F1 (Section 2.3) on the test set.

	H_1	H_2	H_3
BAP-base	11.8	12.4	**14.6**
+ action history	14.6	18.2	**19.7**
+ perspective	15.7	18.7	**18.8**

Table 1: The effect of varying game history and world state representations on test set performance.

	2x	4x	6x
BAP-base$_{H_3}$	15.6	16.1	17.0
+ action history$_{H_3}$	16.9	20.0	18.4
+ perspective$_{H_3}$	19.5	**21.2**	20.8

Table 2: The effect of data augmentation at 2x, 4x and 6x training data on test set performance.

Model Variants The world state representation of the baseline model (BAP-base) consists of block colors at absolute $\langle x, y, z \rangle$ coordinates. We examine the effect of augmenting BAP-base first with action history weights, and then also with relative perspective coordinates (both described in Section 3.3). For model hyperparameters, see Appendix A.

Game History We experiment with three schemes for how much game history to provide to the models: H_1 includes **A**'s last utterance and any following **B** utterances. H_2 includes all utterances after **B**'s penultimate action sequence. H_3 includes all utterances after **B**'s penultimate action sequence interleaved with a token representation of **B**'s last action sequence. If **A**'s last utterance was a standalone instruction, H_1 should be sufficient. But prior discourse is often required: **A** instructions may span multiple utterances and can be interrupted by back-and-forth clarification dialogues. At the same time, **B**'s next action sequence is often directly related to (or a continuation of) their previous actions. This motivates H_2 and H_3: by including utterances that sandwich **B**'s previous action sequence, we include additional **A** history and **B** context. Finally, to investigate the degree to which previous **B** actions should be represented, H_3 augments H_2 with explicit representations of **B**'s actions (as described in Section 3.2).

6 Experimental Results

6.1 Quantitative Evaluation

For each cell in Tables 1 and 2, we first perform a grid search over model hyperparameters and select the best performing model on the development set, then report its performance on the test set.

Table 1 shows how the different game history and world state representations affect model performance. We see that performance increases as action weights are added and as the amount of history is increased. H_3 consistently performs well across all model variants.

Table 2 shows how different amounts of data augmentation affect performance. We train each model variant with H_3 history on 2x, 4x and 6x augmented training data. This increases BAP-base$_{H_3}$'s performance from 14.6 to 17.0 (with 6x data). With action history, performance increases from 19.7 to 20.0. With perspective coordinates, performance increases from 18.8 to 21.2 (both with 4x data). Perspective coordinates, thus, help with more training data (although it is unclear why performance drops again for the more complex models at 6x).

Our best model is the full BAP model with action weights, perspective coordinates, history H_3 and 4x augmented data (BAP$_{H_3,4x}$) with an F1 of 21.2. This is significantly better than the 11.8 F1 of our baseline BAP model with history H_1 and without action history weights, perspective coordinates, or data augmentation (BAP-base$_{H_1}$). We also see an improvement in mean sequence length from 2.23 to 2.66, even if the latter is still much smaller than the mean gold sequence length of 4.3.

Infeasible Actions and Constrained Decoding In any given world state, only a small fraction of the 7623 actions are feasible: blocks can only be placed in locations that are currently empty and adjacent to existing blocks or on the ground, and blocks can only be removed from locations that are currently occupied. Surprisingly, less than 1% of action sequences generated by any of our models contain one or more infeasible actions. We can force our models to predict only feasible actions by multiplying the output of the block action prediction model (post softmax) with a bit mask over block actions that identifies which of the possible actions are feasible in the current world state, but this does not affect the F1 scores of either the baseline model or our best model.

6.2 Qualitative Evaluation

We return to the development set to illustrate different aspects of BAP$_{H_3,4x}$'s generated action sequences. Figures 3 and 4 provide a few examples; more examples can be found in Appendix B.

Colors: Our model is generally able to correctly identify colors of blocks to be placed. While in many cases continuing the color from the previous

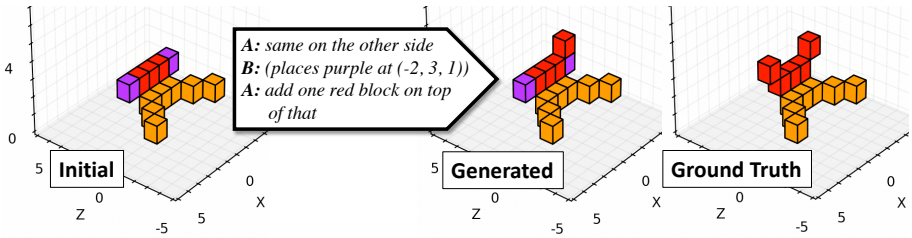

Figure 3: Example 1: After **B** places the rightmost purple block, **A** directs **B** to place another red block on top of it. This occurs after a long back-and-forth clarification dialogue in which **B** struggles to understand **A**'s instructions; but the human **B** now completes the intended substructure by placing two red blocks and removing the purple. The model does not have access to the preceding dialogue, but interprets the most recent instruction correctly.

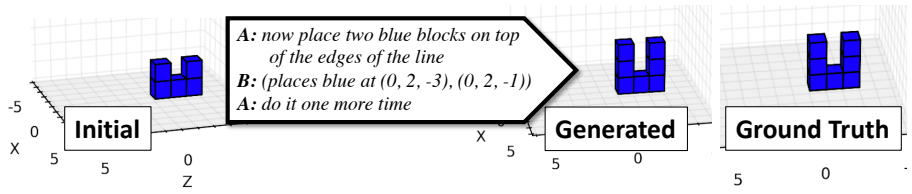

Figure 4: Example 2: Here, **B** had just placed the two blocks atop the ends of the row of 3 blocks to create a *U*. Now, the model can interpret *"do it one more time"* and extends the *U* upwards by placing two more blocks.

action sequence is sufficient, the model is also able to switch colors as needed based on **A** instructions.

Numbers: Our model can sometimes identify the number of blocks to be placed when instructions mention them. But with vague instructions, the model struggles, stopping early or erroneously continuing long sequences of the same color.

Spatial relations: Our model usually predicts a reasonable ballpark of locations for the next action sequence. While predicting correct locations exactly is still difficult, the model is usually able to distinguish *"below"* from *"on top of"*, and places blocks in the neighborhood of the true sequence.

Placements vs. removals: Finally, our model is able to both place and remove blocks somewhat appropriately based on dialogue context. For instance, corrective utterances in the history (*"sorry, my mistake"*) usually trigger the model to undo previous actions. However, the model sometimes goes overboard: not knowing how much of the penultimate action sequence to remove, an entire sequence of correct blocks can be erroneously erased.

7 Conclusion and Future Work

In the Minecraft Collaborative Building Task, Builders must be able to comprehend complex instructions in order to achieve their primary goal of building 3D structures. To this end, we define the challenging subtask of Builder Action Prediction, tasking models with generating appropriate action sequences learned from the actions of human Builders. Our models process the game history along with a 3D representation of the evolving world to predict actions in a sequence-to-sequence fashion. We show that these models, especially when conditioned on a suitable amount of game history and trained on larger amounts of synthetically generated data, improve over naive baselines. In the future, richer representations of the dialogue history (e.g. by using BERT (Devlin et al., 2019) or of past Builder actions) combined with de-noising of the human data and perhaps more exhaustive data augmentation should produce better output sequences. For true interactivity, the Builder must be augmented with the capability to determine when and how to respond when it is too uncertain to act. And, finally, an approach like the Speaker-Follower Models of Fried et al. (2018) could be used to train our Builder model and the Architect model of Narayan-Chen et al. (2019) jointly.

Acknowledgements

We would like to thank the reviewers for their valuable comments. This work was supported by Contract W911NF-15-1-0461 with the US Defense Advanced Research Projects Agency (DARPA) Communicating with Computers Program and the Army Research Office (ARO). Approved for Public Release, Distribution Unlimited. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

References

Peter Anderson, Qi Wu, Damien Teney, Jake Bruce, Mark Johnson, Niko Sünderhauf, Ian D. Reid, Stephen Gould, and Anton van den Hengel. 2018. Vision-and-language navigation: Interpreting visually-grounded navigation instructions in real environments. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 3674–3683. IEEE Computer Society.

Jacob Andreas and Dan Klein. 2015. Alignment-based compositional semantics for instruction following. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1165–1174, Lisbon, Portugal. Association for Computational Linguistics.

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

Yonatan Bisk, Daniel Marcu, and William Wong. 2016a. Towards a dataset for human computer communication via grounded language acquisition. In *AAAI Workshop: Symbiotic Cognitive Systems*.

Yonatan Bisk, Kevin Shih, Yejin Choi, and Daniel Marcu. 2018. Learning interpretable spatial operations in a rich 3D Blocks World. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 5028–5036.

Yonatan Bisk, Deniz Yuret, and Daniel Marcu. 2016b. Natural language communication with robots. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 751–761, San Diego, California. Association for Computational Linguistics.

Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. 2018. MultiWOZ - a large-scale multi-domain wizard-of-Oz dataset for task-oriented dialogue modelling. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 5016–5026, Brussels, Belgium. Association for Computational Linguistics.

Joyce Y. Chai, Qiaozi Gao, Lanbo She, Shaohua Yang, Sari Saba-Sadiya, and Guangyue Xu. 2018. Language to action: Towards interactive task learning with physical agents. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)*, pages 2–9. International Joint Conferences on Artificial Intelligence Organization.

David Chen and Raymond Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 859–865.

Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*.

Abhishek Das, Satwik Kottur, Khushi Gupta, Avi Singh, Deshraj Yadav, José M.F. Moura, Devi Parikh, and Dhruv Batra. 2017. Visual Dialog. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 326–335.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Daniel Fried, Ronghang Hu, Volkan Cirik, Anna Rohrbach, Jacob Andreas, Louis-Philippe Morency, Taylor Berg-Kirkpatrick, Kate Saenko, Dan Klein, and Trevor Darrell. 2018. Speaker-follower models for vision-and-language navigation. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada*, pages 3318–3329.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy. PMLR.

Jonathan Gray, Kavya Srinet, Yacine Jernite, Haonan Yu, Zhuoyuan Chen, Demi Guo, Siddharth Goyal, C. Lawrence Zitnick, and Arthur Szlam. 2019. CraftAssist: A framework for dialogue-enabled interactive agents. *arXiv preprint arXiv:1907.08584*.

Hengyuan Hu, Denis Yarats, Qucheng Gong, Yuandong Tian, and Mike Lewis. 2019. Hierarchical decision making by generating and following natural language instructions. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems*

2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada, pages 10025–10034.

Hsin-Yuan Huang, Eunsol Choi, and Wen-tau Yih. 2019. FlowQA: Grasping flow in history for conversational machine comprehension. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Nikolai Ilinykh, Sina Zarrieß, and David Schlangen. 2019. Meet Up! A corpus of joint activity dialogues in a visual environment. In *Proceedings of the 23rd Workshop on the Semantics and Pragmatics of Dialogue - Full Papers*, London, United Kingdom. SEMDIAL.

Yacine Jernite, Kavya Srinet, Jonathan Gray, and Arthur Szlam. 2019. CraftAssist instruction parsing: Semantic parsing for a Minecraft assistant. *arXiv preprint arXiv:1905.01978*.

Jin-Hwa Kim, Nikita Kitaev, Xinlei Chen, Marcus Rohrbach, Byoung-Tak Zhang, Yuandong Tian, Dhruv Batra, and Devi Parikh. 2019. CoDraw: Collaborative drawing as a testbed for grounded goal-driven communication. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6495–6513, Florence, Italy. Association for Computational Linguistics.

Seokhwan Kim, Luis Fernando D'Haro, Rafael E. Banchs, Jason D. Williams, and Matthew Henderson. 2016a. The fourth dialog state tracking challenge. In *Dialogues with Social Robots - Enablements, Analyses, and Evaluation, Seventh International Workshop on Spoken Dialogue Systems, IWSDS 2016, Saariselkä, Finland, January 13-16, 2016*, volume 427 of *Lecture Notes in Electrical Engineering*, pages 435–449. Springer.

Seokhwan Kim, Luis Fernando D'Haro, Rafael E. Banchs, Jason D. Williams, Matthew Henderson, and Koichiro Yoshino. 2016b. The fifth dialog state tracking challenge. In *2016 IEEE Spoken Language Technology Workshop, SLT 2016, San Diego, CA, USA, December 13-16, 2016*, pages 511–517. IEEE.

Reginald Long, Panupong Pasupat, and Percy Liang. 2016. Simpler context-dependent logical forms via model projections. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1456–1465, Berlin, Germany. Association for Computational Linguistics.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. 2013. Learning to parse natural language commands to a robot control system. In *Proc. of the 13th Int'l Symposium on Experimental Robotics (ISER)*.

Dipendra K. Misra, Jaeyong Sung, Kevin Lee, and Ashutosh Saxena. 2016. Tell me Dave: Context-sensitive grounding of natural language to manipulation instructions. *The International Journal of Robotics Research*, 35(1-3):281–300.

Anjali Narayan-Chen, Prashant Jayannavar, and Julia Hockenmaier. 2019. Collaborative dialogue in Minecraft. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5405–5415, Florence, Italy. Association for Computational Linguistics.

Ramakanth Pasunuru and Mohit Bansal. 2018. Game-based video-context dialogue. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 125–136, Brussels, Belgium. Association for Computational Linguistics.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Alan Ritter, Colin Cherry, and Bill Dolan. 2010. Unsupervised modeling of Twitter conversations. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 172–180, Los Angeles, California. Association for Computational Linguistics.

Deb Roy and Ehud Reiter. 2005. Connecting language to the world. *Artificial Intelligence*, 167(1-2):1–12.

Nicolas Schrading, Cecilia Ovesdotter Alm, Ray Ptucha, and Christopher Homan. 2015. An analysis of domestic abuse discourse on Reddit. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2577–2583, Lisbon, Portugal. Association for Computational Linguistics.

Alane Suhr and Yoav Artzi. 2018. Situated mapping of sequential instructions to actions with single-step reward observation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2072–2082, Melbourne, Australia. Association for Computational Linguistics.

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 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2119–2130, Hong Kong, China. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

Arthur Szlam, Jonathan Gray, Kavya Srinet, Yacine Jernite, Armand Joulin, Gabriel Synnaeve, Douwe Kiela, Haonan Yu, Zhuoyuan Chen, Siddharth Goyal, Demi Guo, Danielle Rothermel, C. Lawrence Zitnick, and Jason Weston. 2019. Why build an assistant in Minecraft? *arXiv preprint arXiv:1907.09273*.

Stefanie Tellex, Thomas Kollar, Steven Dickerson, Matthew Walter, Ashis Banerjee, Seth Teller, and Nicholas Roy. 2011. Understanding natural language commands for robotic navigation and mobile manipulation. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 1507–1514.

Jesse Thomason, Michael Murray, Maya Cakmak, and Luke Zettlemoyer. 2019. Vision-and-dialog navigation. *arXiv preprint arXiv:1907.04957*.

Jesse Thomason, Aishwarya Padmakumar, Jivko Sinapov, Justin Hart, Peter Stone, and Raymond J. Mooney. 2017. Opportunistic active learning for grounding natural language descriptions. In *Proceedings of the 1st Annual Conference on Robot Learning (CoRL-17)*, pages 67–76, Mountain View, California. PMLR.

Jesse Thomason, Jivko Sinapov, Maxwell Svetlik, Peter Stone, and Raymond J. Mooney. 2016. Learning multi-modal grounded linguistic semantics by playing "I Spy". In *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI-16)*, pages 3477–3483, New York City.

Jesse Thomason, Shiqi Zhang, Raymond J Mooney, and Peter Stone. 2015. Learning to interpret natural language commands through human-robot dialog. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 1923–1929.

Sida I. Wang, Samuel Ginn, Percy Liang, and Christopher D. Manning. 2017. Naturalizing a programming language via interactive learning. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 929–938, Vancouver, Canada. Association for Computational Linguistics.

Terry Winograd. 1971. Procedures as a representation for data in a computer program for understanding natural language. Technical report, MIT. Cent. Space Res.

	H	cnn_{type}	l	j
BAP-base$_{H_1}$	300	cnn_{small}	3	1
BAP$_{H_3,4x}$	300	cnn_{small}	4	1

Table 3: Hyperparameter values for the baseline and full BAP models.

A Model Hyperparameters

We use Gated Recurrent Units (GRUs) (Chung et al., 2014) for all RNN modules and use 300-dimensional pretrained GloVe word embeddings (Pennington et al., 2014). All linear layers were initialized using Xavier initialization (Glorot and Bengio, 2010). All non-linearities in the model are ReLU. All $3\times3\times3$ 3d-conv layers have stride 1 and padding 1. All $1\times1\times1$ 3d-conv layers have stride 1 and no padding.

For each model, we perform a grid search over the following hyperparameters:

- The size of the GRU hidden state $H \in \{200, 300\}$
- The number of 3d-conv layers and channels in the world state encoder and action sequence decoder CNNs. We define a 3-tuple ($e_{channels}$, m, n) where $e_{channels}$ defines the number of output channels for the first encoder-CNN 3d-conv layer (which then determines the number of output channels for subsequent encoder-CNN 3d-conv layers); m is the number of $3\times3\times3$ 3d-conv layers in the world state encoder; and n is the number of $1\times1\times1$ 3d-conv layers in the action sequence decoder.
  We choose between 2 hyperparameter configurations: $cnn_{type} \in \{cnn_{small} = (200, 2, 3), cnn_{big} = (300, 3, 2)\}$.
- The number of dense linear layers in the STOP prediction model $l \in \{3, 4\}$
- The number of dense linear layers used to embed the action vectors before being fed to the decoder's GRU $j \in \{1, 2\}$

Table 3 shows values of these hyperparameters for our baseline and best models.

B Qualitative Examples

Here, we provide more examples of action sequences generated by our model, along with the initial game state context and the human **B**'s actions as ground truth, in order to better highlight

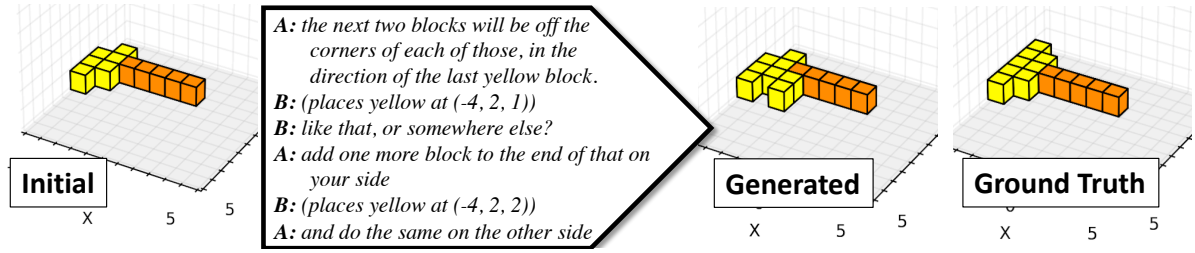

Figure 5: Example 3.

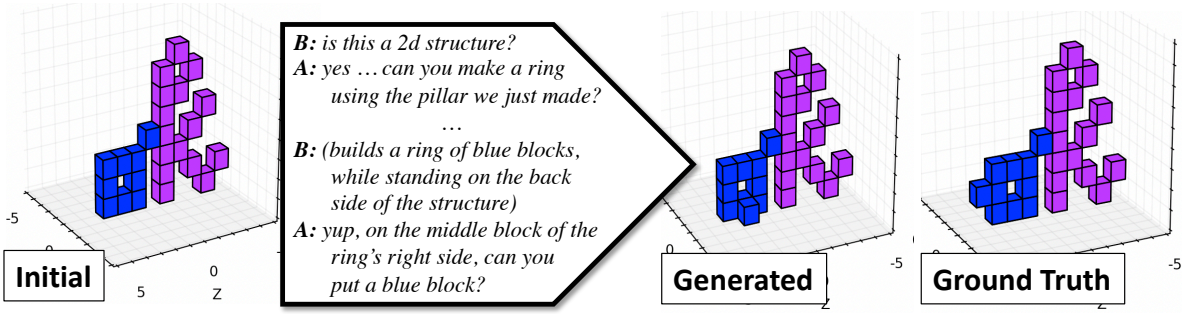

Figure 6: Example 4.

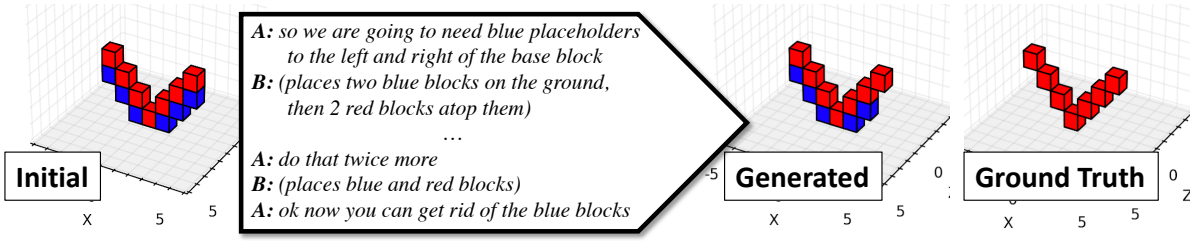

Figure 7: Example 5.

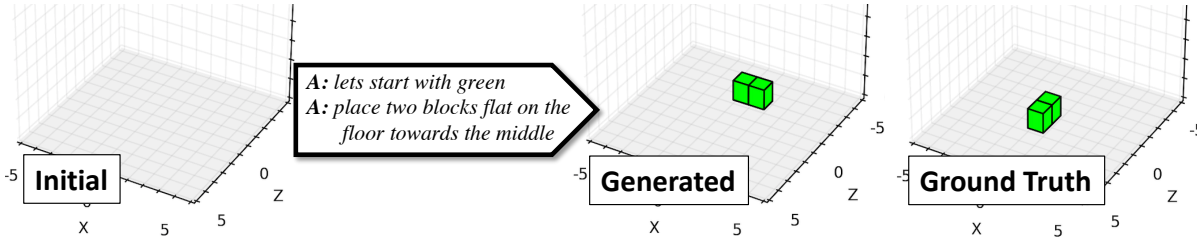

Figure 8: Example 6.

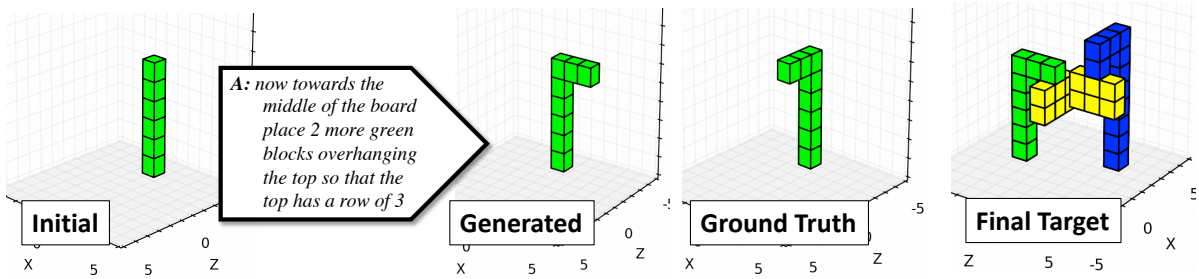

Figure 9: Example 7.

the strengths and shortcomings of the full BAP model. Examples 5, 6 and 7 also examine the net actions F1 evaluation metric in context.

Example 3 can be found in Figure 5. Over the course of some back-and-forth dialogue with **A**, **B** has just built the leftmost 2 yellow blocks of the left yellow row. From here, our model interprets *"do the same on the other side"* as placing another 2 yellow blocks, but places them in the wrong location. The human **B** is able to understand that **A** means to place the blocks on the other end of the row-in-progress.

Example 4 can be found in Figure 6. This example occurs near the end of a game. **B** has just finished building a 3×3 ring of blue blocks, while facing the structure from the back side (i.e., facing the camera in the figure). Following the description *"the middle block of the ring's right side"*, our model incorrectly predicts placing a blue block adjacent to one of the *middle* blocks of the ring, while the human **B** grounds this easily. Clearly, higher-level information needed to help ground the instruction is lost in context: earlier in the dialogue history (yet still within the window of utterances in the H_3 history scheme), **B** has clarified with **A** that the structure is entirely 2D, which contradicts the model's prediction.

Example 5 can be found in Figure 7. **B** has built a *V* using blue blocks as placeholders to support the red blocks. Our model interprets *"get rid of the blue blocks"* partially correctly, and removes one blue block, but does not go all the way as the human **B** does, who removes all existing blue blocks. While both the model's and human **B**'s action sequences are correct, the model's actions are incomplete, and it is penalized according to net actions F1.

Example 6 can be found in Figure 8. This example occurs at the beginning of a game. Here, **A** does not specify a specific location for the green blocks to be placed, just that they should be *"towards the middle."* In this instance, both our model's prediction and the human **B**'s actions are valid interpretations. However, our model's output is penalized for not predicting the exact positions of the human **B**'s blocks. This highlights the net actions F1 metric's inflexibility to ambiguous scenarios.

Example 7 can be found in Figure 9. This example is similar to Example 8 in that the model predicts a sequence of actions that results in a structure that is rotationally equivalent to the human **B**'s resulting structure. However, in this case, **A**'s instruction to place the green blocks *"towards the middle of the board"* (a suggestion our model does not follow) is extremely important in the larger context of task completion: the model's actions would result in a final structure that cannot fit within the grid boundaries. Here, the strictness of net action F1's exact match requirement works as intended, to our benefit.