From Words to Blocks: Building Objects by Grounding Language Models with Reinforcement Learning

Anonymous ACL submission

Abstract

Leveraging pre-trained language models to generate action plans for embodied agents is an emerging research direction. However, executing instructions in real or simulated environments necessitates verifying the feasibility of actions and their relevance in achieving a goal. We introduce a novel method that integrates a language model and reinforcement learning for constructing objects in a Minecraft-like environment, based on natural language instructions. Our method generates a set of consistently achievable sub-goals derived from the instructions and subsequently completes the associated sub-tasks using a pre-trained RL policy. We employ the IGLU competition, which is based on the Minecraft-like simulator, as our test environment, and compare our approach to the competition’s top-performing solutions. Our approach outperforms existing solutions in terms of both the quality of the language model and the quality of the structures built within the IGLU environment.

1 Introduction

The ability to solve complex tasks, formulated in natural language, that require a long sequence of actions in the environment is a fundamental property of human intelligence. The recent significant success of large language models (LLMs) in instruction following, explanation generation, and planning demonstrates their powerful capabilities in solving commonsense, general knowledge, and code generation problems within the verbal domain. However, LLMs are trained solely on textual data, which limits their ability to understand and perform actions in real-world-like environments. Consequently, even ChatGPT (Ouyang et al., 2022) exhibits poor spatial reasoning (Bang et al., 2023).

On the other hand, reinforcement learning (RL) has proven effective in learning sequences of fine-grained actions for specific tasks within an environment. Thus, investigating the combination of LLMs for natural language understanding and high-level planning, along with RL for learning environmental manipulation, represents a promising research direction.

LLMs can be regarded as universal knowledge bases that allow human users to interact in natural language and solve complex tasks (Huang et al., 2022; Zeng et al., 2022). Recent studies have shown that pre-trained LLMs can construct high-level action plans in both simulated and real environments (Bara et al., 2021; Min et al., 2022; Murray and Cakmak, 2022). However, these LLM-based approaches necessitate manual prompt engineering, handcrafted translation of language commands into embodied actions, and a strategy for goal-aware action selection from the distribution of potential options generated by language. In this context, several studies (Ahn et al., 2022; Ouyang et al., 2022) have demonstrated that the rough action plans extracted from language models can be refined using reinforcement learning (RL).

In this paper, we propose a method that utilizes a pre-trained LM, without the need for prompt engineering, to learn a language-conditioned multitask RL policy for executing natural language instruc-
tions to construct various figures in a Minecraft-like environment (Kiseleva et al., 2022a). We employ a simple embodied environment (Zholus et al., 2022), introduced as part of the IGLU competition, in which a Builder agent must assemble a structure described by natural language instructions provided by an Architect agent (see an example in Figure 1).

Our Language to Subtask Builder (L2S) is composed of three main modules. In inference mode, the process begins with the Language Module, which translates a natural language instruction into a sequence of block coordinates to be placed within the environment. Subsequently, the Task Manager converts this sequence into an ordered list of sub-goals for execution. Lastly, the Policy Module employs a pre-trained reinforcement learning (RL) based policy to accomplish each sub-goal, while simultaneously monitoring the success of individual actions.

We present a novel method for decomposing the initial task into separate, independent subtasks to address the known limitations of the plan length generated by the LLM. Executing these subtasks in any order enables the L2S Builder to significantly enhance the quality of partial plans produced by the LLM, resulting in improved performance in both the prediction of the figure structures described in the instructions and the final structural metric in the IGLU competition. Furthermore, we propose several techniques for processing instructional data to bolster the model’s generalization capabilities. These approaches center around augmenting the original dataset by employing prompt queries to a ChatGPT model. We provide the code and data in the anonymized repository.

2 Related work
Recent studies are actively exploring the generation of action plans using language models. Some works focus on prompt engineering for effective plan generation (Singh et al., 2022), while others address the challenge of translating the language model’s output into executable actions (Huang et al., 2022). In (Ahn et al., 2022), models are trained with reinforcement learning (RL) for selecting feasible actions and executing elementary actions. Many of these works (Shridhar et al., 2020; Lynch et al., 2022) aim to control robots interactively in real time. Maintaining a dialogue with humans is an essential area of research for robotics (Padmakumar et al., 2022; Gao et al., 2022).

Designing an effective learning process and providing sufficient training data are the main problems in building interactive embodied agents that can learn to solve tasks using grounded natural language instructions in a collaborative environment. In the paper (Narayan-Chen et al., 2019), the authors present a Minecraft-like environment and a dataset containing interactions between a human architect and builder. The architect knows the structure that needs to be built and has to explain how to construct it to the builder using natural language. The paper primarily focuses on developing and evaluating a model for predicting the architect’s instructions. The work has been extended in another paper (Shi et al., 2022), which annotates all builder utterances into eight types, including clarification questions. Based on the annotated data, the proposed builder agent model can determine when to ask for clarification or execute instructions. The authors of the IGLU competition (Kiseleva et al., 2022a,b) expanded the dataset and provided a baseline for constructing figures in an embodied setting. This baseline uses a vector of environment features as observation, including the agent’s position and directions and the positions of the blocks that make up the figure.

The paper (Bonial et al., 2021) proposes a natural language understanding pipeline to translate free-form natural language input into executable actions, consisting of two steps: Abstract Meaning Representation (AMR) parsing and conversion into Dialogue-AMR. The task of predicting the builder’s action sequences, using only block placements and removals actions (no embodiment), is considered in the paper (Jayanavar et al., 2020). A similar problem was solved in the work (Barde et al., 2022), which proposes a new architecture for the simultaneous training of an architect and a builder in a toy grid-based environment.

The human-guided collaborative problem-solving framework (Kokel et al., 2022a,b) proposes a nonparametric method that utilizes rule-based primitives and hierarchical task network (HTN) planners to construct structures. The use of HTN planners for predicting building behavior is not a new concept, as it has already been employed in previous work (Köhnl et al., 2020), which divided
the goal space into several levels of abstraction. In that case, the goals were subtasks for constructing specific parts of a larger figure. In contrast, our proposed approach involves fine-tuning the language model, decomposition into subtasks independent of the sequence of their execution, and pre-trained RK-based navigation skills that ground that plan into the environment.

3 Problem setting

3.1 Environment

For our task, we use IGLU environment\(^3\) where an embodied agent can build spatial structures of blocks (figures) of different colors. The agent’s goal is to complete a task expressed as an instruction written in natural language.

The observation space consists of point-of-view (POV) image \((64, 64, 3)\), inventory item counts \((6)\), a snapshot of the building zone \((11, 11, 9)\), and the agent’s \((X, Y, Z)\) position with pitch and yaw angles \((5)\). The building zone is represented via a 3D tensor with block ids (e.g., 0 for air, 1 for blue block, etc.). The agent can navigate over the building zone, place, and break blocks, and switch between block types.

Additionally, the environment provides a dialog from the dataset, which defines a building task. The dialog is split into two parts: the first is a context for the task, and the second defines the target. The context utterances define blocks that are placed before the beginning of the episode. We thus initialize the episode with such blocks placed at their correct positions. Target utterances define the rest of the blocks needed to be added. The environment provides two modes of action for choice: walking and flying. We use the flying mode, which in contrast with walking, does not have gravitation-affecting agents, but the movement actions are continuous.

The action space combines discrete and continuous subspaces: a discrete space of 13 actions (noop, four camera rotation actions, break block, place block, choose block type 1-6) and a set of 6 continuous actions for movement in all three directions.

The reward in the environment reflects how to complete the built structure is agnostic to its location. We run a convolution-like procedure between the state grid and the target grid. We calculate the per-block intersection for each alignment and then maximize it over alignments.

3.2 Evaluation Metrics

In the competition, two metrics were proposed. The first metric, \(F_1\), is based on finding the maximum intersection of 3D voxels between the original structure and the structure created by the agent. The second metric, Strict \(F_1\), penalizes the structure if it is not built in the direction described in the dialogue.

We have developed a metric called Voxel Structural Similarity (VSS) to evaluate the structural similarity between two voxel structures. VSS captures spatial relationships between voxels by converting each structure into an ordered set of structured blocks. Each structured block is a distance-sorted list of other blocks relative to a target block, encoded with angle and color.

When computing the metric, we match the blocks in the original and predicted structures and assign a similarity value to each pair. We find the best matches using the Hungarian algorithm and average their similarity values.

Unlike \(F_1\), which considers the maximum sequence of similar blocks and may overlook visually significant information, VSS considers not only the absolute position of each block but also their interrelation within the structure. Therefore, if some blocks are correctly positioned relative to each other but have minor errors in absolute positioning, VSS may evaluate the model higher than \(F_1\) since it considers the visual perception of the structure as a whole rather than its elements.

In our perspective, we assert that an optimal evaluation framework ought to encompass all three metrics to attain a comprehensive comprehension of the agent’s performance. In pursuit of this objective, the integration of a composite metric, such as the Harmonic Mean (HM), which amalgamates all three metrics, could provide an equitable assessment of the agent’s performance. The Harmonic Mean, characterized by its responsiveness to low values, guarantees that a superior score can only be attained through commendable performance across all metrics. Appendix A provides a detailed description of the VSS calculation.

4 The Language to Subtask Builder (L2S)

Training an agent to construct an arbitrary structure described in natural language poses significant challenges. Even if the task is directly given to the agent by specifying target positions of elements on a grid, the problem remains nontrivial.

\(^3\)https://github.com/iglu-contest/gridworld
Figure 2: Diagram of Language to Subtask Builder (L2S). The agent consists of three parts. First, the Language module translates natural language instructions into environmental block coordinates. Next, the Task Manager converts these coordinates into an ordered sub-goal list. Lastly, the Policy module executes a pre-trained RL-based policy to complete each sub-goal while monitoring the success of actions.

We introduce an approach for training a general-purpose builder, capable of solving structures not encountered during the training phase. The Language to Subtask Builder (L2S) agent converts the natural language description of a target structure into a grid representation, which defines subtasks that are sequentially completed by an RL policy (see Figure 2).

The process begins with the prediction of which blocks must be placed based on the given instruction. Subsequently, the agent performs a series of actions in the environment, leading to block placements in accordance with the instruction. The builder agent comprises three modules:

1. the Language module, which predicts the coordinates and types of blocks from a given text,
2. the Rule-based Task Manager, which iterates over the positions of predicted blocks in a heuristically defined order, and
3. the RL-based Policy module, responsible for solving atomic subtasks, such as individual block placement or removal.

The RL module operates using visual input, inventory, and compass observation provided by the environment, along with the target block position supplied by the Task Manager.

4.1 Language Module

Given an instruction as input the Language model is expected to output a sequence of subtasks in form of a string. Each subtask has a form of tuple of coordinates and color: (z, x, y, color). The output of the model is some number of these tuples separated by commas. The resulting string is converted to 3-dimensional array of predicted voxel and is then sent to the next module.

4.2 Task Manager

This module translates the voxel representation predicted by the Language module into a sequence of simple subtasks that require either placing or removing a single block. Subtasks are issued in a certain sequence, from bottom to top, from left to right, so when building a structure, the agent never encounters obstacles, which further simplifies the learning process.

However, some structures may require building "flying" blocks i.e., blocks that have no supporting block below them. To handle that the module first generates auxiliary subtasks for building blocks below the target block. After the target block is placed the module generates subtasks for removing redundant blocks below it.

4.3 Navigation Module

Reaching a subtask in the environment is a sequential decision-making problem. As it requires making decisions based on visual input and quantitative assessment of the quality of the achieved result, we use reinforcement learning as it is considered the best tool for that type of problem. In reinforcement learning, there is an agent’s policy $\pi_\theta(s_t)$ which
outputs an action (or a distribution over the set of available actions) given the current state $s_t$ of the environment. The policy’s parameters $\theta$ are optimized to maximize a sum of scalar rewards $r$, which are given as feedback from the environment at each step, over an episode. As the choice for the agent’s model architecture is arbitrary, we use a model with a recurrent neural network in it so the agent will consider not only the current observation $o_t$ (POV) at each step $t$ but also previous observations to approximate a real environment state $s_t$. We use high-performant implementation (Petrenko et al., 2020) of Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017).

The Navigation module is responsible for implementing a single subtask in the environment guided by POV. It is used iteratively for all the subtasks from the Task Manager, and after each subtask, the agent’s recurrent hidden state is reset. In the pre-trained phase of this module, the reinforcement learning agent learns to navigate to a specified block within a subtask from any position. It is trained with reduced action space containing only actions for movement, camera rotation, and action done, which indicates that the agent finished the subtask. The agent has a wrapper that translates the agent’s actions into the environment. After the agent takes the action done the wrapper decides whether to break or to put the block and in the former case it chooses the color. The information about the type of action and the block’s color is available from the subtask itself. Therefore we say that the RL agent only navigates to the block while the action to be taken is predicted by the Language model. It may happen that the agent wants to place a block, but the distance to the ground/another block is too large so the action cannot be implemented. The subtask is considered as completed only after it is implemented in the environment and not after the action done was taken. The wrapper knows the environment changed from the inventory: it decreases if a block is placed and increases if broken. So if the agent took the done action, but the inventory did not change, the wrapper sends the same subtask to the agent again while it is not completed properly.

5 Experiments

5.1 Baselines

We have selected the three best solutions from the IGLU 2022 competition for comparison. T5 (Raf-

5.2 Language Module

5.2.1 Dataset

The train and test datasets contain 109 and 41 instructions in the English language with corresponding grids to be built. Each instruction has a division for steps of construction with a corresponding grid for each of them. The grids are converted to a sequence of coordinates and colors of blocks in form of a string of tuples. If after implementing a step of instruction some blocks were removed, the tuples for these blocks will have word remove in the place of the color. The order of the blocks in the sequence corresponds to the appearance of the blocks in the grid with the following signs of directions: first along $x$, then $y$, and then $z$. A lot of instructions do not specify the position of the first block and it can appear anywhere in the grid along $x$ and $y$ randomly. It is harmful to the model during training because it will receive a loss for tokens that cannot be deduced from the input. Therefore we change the sequence so that the first block of the first step of instruction will be placed at the center of the space with the remaining blocks shifted appropriately.

5.2.2 Zero-Shot learning on IGLU Dataset

As part of an experiment, we explored the use of zero-shot learning and prompts with the ChatGPT model to predict the coordinates required for setting in the environment. We formulated prompts $B$ that describe the environment in which ChatGPT needs to construct shapes. We described a coordinate grid and how the coordinates increase relative to the cardinal directions. We provided certain vocabulary patterns commonly encountered in the environment (row, column, tower) that specify the positions of the primitives in the data. Subsequently, within a single session, we sequentially issued instructions for one shape, where the task

4https://github.com/kakaobrain/brain-agent
for ChatGPT was to understand the actions to be performed (Table 2) within the instructions. The analysis of the results demonstrates the potential outcomes achievable with our approach by enhancing the language model.

5.2.3 Strategies for Mitigating Overfitting for IGLU Dataset

The limited size and unclean nature of the original dataset constrain the language model’s performance. To overcome overfitting and increase representativeness, we implemented three strategies to enhance the model’s quality.

A. Color Augmentation

For every element in the dataset, we generate two more elements with the same instruction and target but with blocks of different colors. This is done by simply replacing color names in the instruction with other colors and appropriately changing the colors of the blocks in the target grid. This augmentation is applied before the training to all the datasets including datasets produced by the following augmentations.

B. Rephrasing Text Instructions

To improve the model’s understanding of the various ways in which humans describe shapes, we used ChatGPT to rephrase the original text instructions C. This approach provided the model with additional variations of the same instructions and helped it learn to recognize patterns and understand complex instructions.

C. Updating Coordinates

To improve the model’s understanding of cardinal directions, we displaced the shapes along different axes and asked ChatGPT to generate new instructions for each shape based on the updated coordinate system C. This approach helped the model learn to generalize better to different orientations of the same shapes.

5.2.4 Figure decomposition

Additionally, we propose a method of decomposing shapes into primitives that need to be constructed. Upon careful examination of the data, it can be noticed that each individual instruction suggests building either a primitive (i.e. a three-dimensional rectangle of specified dimensions), removing certain blocks from the shape, or placing blocks on top of specific existing ones. If we only consider instructions that require the construction of primitives, the maximum F-score of this dataset will be 0.7.

Thus, instead of predicting an instruction from blocks of unlimited length, we can predict a list of three tuples, where the first tuple is the coordinate point where the primitive should be placed (closest to the origin of the shape), the tuple of width, length, and height dimensions, and a conditional representation of the color of the primitive that needs to be placed.

Thus, for instructions like: “Facing North, build a flat square of orange blocks of 4 x 4 blocks in the middle of the grid.” The prediction will not be a long list of 16 coordinates, but one short list of the form: 

\[
(0, 5, 5), (0, 4, 4), 'orange'
\]

5.2.5 Training

We use encoder-decoder transformer model Flan-T5 (Chung et al., 2022) as the language model. For comparison of the augmentation strategies we use 220 million parameters Flan-T5-base. For the final evaluation we use 770 million parameters Flan-T5-large. The encoder-decoder architecture best fits the problem because it can be stated as a translation from a free language figure description to a fixed format description, and there is a direct correspondence between output and input tokens. Also, in the dataset, the input utterances have a form of instructions. Therefore, we use the Flan-T5 model, which was trained on instructional data. The models are trained in seq2seq manner converting the instruction into a sequence of blocks as described above with batch size 8, learning rate 0.00001 for 80000 training steps. It took 12 hours on Titan RTX GPU.

5.3 RL Training

The RL agent learns independently from the NLP model, and its tasks are procedurally generated. An episode in the environment starts with some blocks already built, and the task within one episode is either to place or destroy one block. The blocks built at the start of the episode are randomly generated with a probability of 0.7, while the blocks are selected from grids in the training dataset with a probability of 0.3. The agent’s objective is to navigate to the target block in the environment, and upon reaching it, the agent receives a reward based on the degree of error in its position selection. The reward function is defined based on the Manhattan distance between the placed block and the target position, replicating the reward function of the IGLU baseline.

We use a high-throughput implementation of
PPO for policy optimization. The policy network uses a residual convolutional encoder to encode the input, reproducing the backbone of IMPALA architecture (Espeholt et al., 2018) for processing RGB images. We use the same architecture removing max-pooling layers to encode the target block. The full architecture is present in Appendix D. The output of these encoders was concatenated with additional environment information (compass and inventory) processed by multilayer perceptron (MLP) layer. The resultant vector is passed to the long short-term memory (Hochreiter and Schmidhuber, 1997) (LSTM) head. The used hyperparameters are presented in Appendix E.

We trained the RL policy for 175 million environment steps, it took 6 hours on Titan RTX GPU. The success rate of subtask completion achieved 0.95.

6 Results and Discussion

The training of the baseline agent used in the IGLO competition motivates the agent to place cubes directly under itself, making its behavior more controllable, but also limiting the possibility of more interesting strategies, such as building blocks adjacent to the figure.

By narrowing the agent’s action space, removing the action of removing and adding cubes (this process is delegated to an NLP model and a task manager), and changing the environment’s action space to flying, the agent no longer needs to build support blocks adjacent to other figures, which reduces the possibility of making mistakes in building the complete figure and reduces the number of actions needed to build it.

As a result, the final agent achieved a significantly higher SR metric result (0.95) than the agent used in the baseline (0.85).

We compared NLP model training for different data processing strategies (Table 1). The first strategy (A) involved increasing the original dataset by a factor of 3 by replacing colors with commands and figures. Strategy (ABC) included color augmentation as well as augmenting the direction of light in dialogues (rotating figures by 180 degrees) and rephrasing those dialogues.

During the experiment, we noticed that simple expansion of the original dataset did not lead to a significant performance improvement. Therefore, we also decided to try training the model not on the full set of augmented data, but to use stratified subsets from data A (strat) and ABC (strat). At each epoch of training, we sampled a fixed amount of data (equal to twice the amount of data in the real dataset) from the original and augmented sets.

Thus, in the case of A(strat), we sampled from the original dataset and the color-augmented dataset, and in the case of ABC (strat), we sampled from the original dataset and the ABC-augmented datasets.

The fifth strategy (prim) is a change in the dataset format, replacing the answer in the form of coordinates with an answer in the form of a primitive description 5.2.4. For training on primitives, we did not use any additional augmentation, meaning we only used dialogues from the training dataset without any modifications.

<table>
<thead>
<tr>
<th>data</th>
<th>A</th>
<th>ABC</th>
<th>A(strat)</th>
<th>ABC(strat)</th>
<th>prim</th>
</tr>
</thead>
<tbody>
<tr>
<td>VSS</td>
<td>0.24</td>
<td>0.28</td>
<td>0.26</td>
<td>0.27</td>
<td>0.32</td>
</tr>
<tr>
<td>F1</td>
<td>0.42</td>
<td>0.45</td>
<td>0.43</td>
<td>0.48</td>
<td>0.5</td>
</tr>
<tr>
<td>Strict F1</td>
<td>0.35</td>
<td>0.38</td>
<td>0.35</td>
<td>0.42</td>
<td>0.43</td>
</tr>
<tr>
<td>HM</td>
<td>0.32</td>
<td>0.36</td>
<td>0.33</td>
<td>0.37</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Table 1: Data augmentation strategies. This table compares the F1 score of the FlanT5-base model trained on two augmentation strategies. Strategy A involved replacing colors, while Strategy ABC involved replacing colors and rotating figures. The A(strat) and ABC(strat) datasets were used for epoch sampling using stratification.

Based on the results, we hypothesize that training solely on augmented data (strategies ABC and A) helps the model better learn to identify patterns in dialogues and relate them to the shapes that need to be constructed (e.g., row, line, square, etc.). Furthermore, increasing the use of colors improves the construction of these patterns using specific color associations.

At the same time, restructuring the data into a primitive format helps establish clearer connections between the patterns present in the instructions and the actual shapes. In a way, it simplifies the task as it becomes a classification problem—determining which pattern needs to be constructed from the instructions, identifying its rotation and color. The most challenging aspect for the agent remains determining the position where the pattern needs to be placed. From the table, it is evident that despite removing a portion of the instructions related to removing or constructing complex shapes in a single iteration, the prediction quality is higher compared to agents predicting coordinates.

However, because color augmentation generates
## Text to Blocks Prediction

<table>
<thead>
<tr>
<th></th>
<th>T5 (MHB)</th>
<th>Pegasus</th>
<th>BrainAgent</th>
<th>L2S (coords)</th>
<th>L2S (prim)</th>
<th>L2S (Chat-GPT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NLP VSS</td>
<td>0.14</td>
<td>0.14</td>
<td>-</td>
<td><strong>0.36</strong></td>
<td>0.32</td>
<td>0.31</td>
</tr>
<tr>
<td>F1</td>
<td>0.37</td>
<td>0.36</td>
<td>-</td>
<td><strong>0.59</strong></td>
<td>0.5</td>
<td><strong>0.63</strong></td>
</tr>
<tr>
<td>Strict F1</td>
<td>0.32</td>
<td>0.31</td>
<td>-</td>
<td><strong>0.51</strong></td>
<td>0.43</td>
<td>0.45</td>
</tr>
<tr>
<td>HM</td>
<td>0.23</td>
<td>0.23</td>
<td>-</td>
<td><strong>0.47</strong></td>
<td>0.4</td>
<td>0.43</td>
</tr>
</tbody>
</table>

## IGLU Environment

<table>
<thead>
<tr>
<th></th>
<th>T5 (MHB)</th>
<th>Pegasus</th>
<th>BrainAgent</th>
<th>L2S (coords)</th>
<th>L2S (prim)</th>
<th>L2S (Chat-GPT)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VSS</td>
<td>0.06</td>
<td>0.04</td>
<td>0.21</td>
<td><strong>0.31</strong></td>
<td>0.25</td>
<td><strong>0.33</strong></td>
</tr>
<tr>
<td>F1</td>
<td>0.33</td>
<td>0.26</td>
<td>0.45</td>
<td><strong>0.52</strong></td>
<td>0.43</td>
<td>0.49</td>
</tr>
<tr>
<td>Strict F1</td>
<td>0.29</td>
<td>0.21</td>
<td>0.31</td>
<td><strong>0.48</strong></td>
<td>0.39</td>
<td>0.42</td>
</tr>
<tr>
<td>HM</td>
<td>0.13</td>
<td>0.09</td>
<td>0.3</td>
<td><strong>0.42</strong></td>
<td>0.34</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Table 2: The table shows the mean scores of metrics for all test tasks in the IGLU dataset calculated on 5 seeds. T5 uses a 770M parameter T5-large model with an RL agent from the IGLU public baseline. Pegasus uses a 568M parameter Pegasus-large model with an RL agent from the IGLU public baseline. BrainAgent is an end-to-end model with 640M parameters without division into NLP and RL modules. VSS, F1, and Strict F1 are metrics used during testing, and HM represents their harmonic mean. The best performing approach is highlighted in bold, and the best performing model, including proprietary models (i.e. Chat-GPT), is highlighted in pink color.

...
ity improvement for this approach when adding augmented data or using the more powerful FlanT5 Large model. Nonetheless, it can be observed that after constructing shapes predicted by primitives, the metrics decrease less sharply compared to constructing shapes based on coordinate predictions. These two observations suggest that the figures predicted by our decomposition approach have fewer errors during construction. Additionally, since the shape to be constructed is divided into rectangles, it is easier for the agent to construct it compared to a set of randomly scattered blocks.

7 Conclusions

This paper presents the Language to Subtask Builder (L2S) approach for converting natural language instructions into a sequence of block coordinates and constructing 3D voxel structures in an environment. Our approach consists of three main modules: the Language module, the Task Manager, and the Policy module.

Presented L2S approach surpasses IGLU-2022 solutions in all presented metrics, excelling in deducing the figure construction direction from dialogue context. Comparing the approach based on the prediction of the coordinates sequence and based on the prediction of primitives, we can say, that the L2S (coords) method builds complex structures with correct relative block placements but some dispersion, while the L2S (prim) approach predicts more accurately but may result in larger VSS metric errors due to small initial position or pattern size errors causing significant structural changes. It is worth noting that L2S (prim) shows its potential by achieving improvement simply through changing the output format, as evident from the table 1.

Despite focusing only on one task of the IGLU competition in our work, we propose that our approach can serve as a fundamental basis for instructional reinforcement learning tasks, such as MINE DOJO and Basalt.

We explicitly decompose the task of embodied learning into two components: an NLP module that translates textual commands into subtasks that can be performed by an RL agent, and a navigation module trained using RL strategies. This approach allows both components to be trained independently, providing additional freedom for tuning these models. For the NLP module, this includes data augmentation and transformation, while for RL, it involves flexible hyperparameter tuning.

Another advantage of such decomposition is the ability to replace one of the components without the need to retrain all the modules. Thus, in a real-world scenario, the NLP module can be substituted with a more powerful one, such as ChatGPT.

As a potential improvement, we envision modifying the loss function of the language model by incorporating the target metric we want to optimize, such as F1 or VSS.

8 Limitations

One limitation of our work is the absence of a feedback loop between the agent’s observations and the NLP model’s decision-making. For instance, if an agent starts operating in a world where something has been previously constructed (e.g., by another agent) or failed to build the exact figure they requested the architects to build, they cannot communicate this information.

We incorporated API access to the proprietary ChatGPT model to supplement IGLU dataset, in addition to using other models and freely licensed code in our work. It’s important to note that the lack of high-performing and instructive language models could potentially impede researchers who aim to replicate this augmentation technique.

In our study, a separate question is how to search for the most representative metrics for comparing two volumetric structures that best correlate with human visual perception. We can pinpoint several limitations by examining the metrics presented in the study and the IGLU competition. For instance, the proposed F1 metric only considers the optimal intersection between shapes, neglecting information about other blocks in the constructed shape. These blocks might have errors but are correctly positioned relative to the overall shape. Consequently, a shape with a better visual similarity may have lower metrics than one with a superior intersection with the target.

Another debated aspect is whether the evaluation should consider the shape’s rotation accuracy based on the dialogue description. If an agent can build a shape according to the directions in the dialogue, it suggests improved comprehension of instructions involving cardinal directions. However, if the agent can construct the correct shape without relying on the direction, it implies an understanding of block arrangement relative to each other, even when commands come from a different perspective.
References


So Yeon Min, Devendra Singh Chaplot, Pradeep Ravikumar, Yonatan Bisk, and Ruslan Salakhutdinov. 2022. FILM: Following Instructions in Language with Modular Methods. In ICLR.


A Structural Similarity Metric Calculation

The Voxel Structural Similarity (VSS) captures the spatial relationships between voxels by converting each voxel structure into an ordered set of structured blocks. Each structured block is formed by comparing it to all other voxels within the same structure, capturing their color, angle, and distance information. The angle is calculated as the difference in angles in the xz-plane and xy-plane, providing a measure of the relative orientation of the voxels. The distance between voxels is computed using the Euclidean distance, reflecting their spatial separation.

![Figure 3: Changing the metric depending on the error made.](image)

Unlike metrics that only focus on the maximum possible intersection between two structures, this algorithm evaluates the dependencies and relative positioning of blocks within each structure. On the figure, you can see how the value of the metric changes depending on the error made when building the figure. Consequently, it is less sensitive to...
small errors, such as when an agent places a block slightly further away from the target. This tolerance to minor deviations leads to a more accurate assessment of the structural similarity that closely resembles human perception.

Humans tend to assess similarity by considering not only the absolute positions of individual elements but also their relative arrangement within a structure. By incorporating the dependencies and relative positioning of blocks in the metric, the VSS algorithm mimics this aspect of human perception, providing a more intuitive and meaningful measure of structural similarity.

This human-centric evaluation is particularly beneficial in applications where the goal is to create structures that visually or functionally resemble a target, as it ensures that the resulting structures align with human expectations and preferences.

Steps for metric calculation:

1. We use special data structure VoxelStruct, which takes a voxel structure and converts it into an ordered set of structured blocks:

   \[ \text{VoxelStruct}(\text{voxel}) = \{ s_1, s_2, \ldots, s_n \} \]

   Here, each structured block is an ordered set of elements of the form:

   \[ s_i = \{ (c_1, \theta_1, d_1), (c_2, \theta_2, d_2), \ldots \} \]

   where \( c \) is the color, \( \theta \) is the angle, and \( d \) is the distance between blocks.

2. The function \_get_max_subsequence_length calculates the maximum length of the subsequence (LCS) for a pair of structured blocks:

   \[ \text{LCS}(s_i, s_j) = \max_{1 \leq k \leq |s_i|, 1 \leq l \leq |s_j|} \{ \text{LCS}(s_i[1 : k], s_j[1 : l]) \} \]

3. The function compare_voxel_structs compares two voxel structures and calculates their similarity. First, a similarity matrix is created:

   \[ S_{ij} = \frac{\text{LCS}(s_i, s'_j)}{a} \]

   where \( s_i \) is a structured block from the first structure, \( s'_j \) is a structured block from the second structure, and \( a \) is the number of structured blocks in the first structure.

4. Then, the Hungarian algorithm is used to find the optimal set of correspondences between structured blocks:

   \[ \text{best_matches} = \text{linear_sum_assignment}(-S) \]

5. Finally, the overall similarity measure is calculated based on the N best matches:

   \[ \text{similarity_measure} = \sum_{(i,j) \in \text{best_matches}} S_{ij} \]

**B  Coordinates Generation Prompt**

First, we include a description of the IGLOO environment in the prompt. This way, we provide information on how the coordinates change in relation to cardinal directions (north, west, sky). We also provide information about certain patterns present in the dataset (rows, squares, columns, towers) and what they signify in the dataset.

*Environment have size (11 1 1 9). Coordinates are (x y z). x coordinate increases towards south. y coordinate increases towards east. z coordinate increases towards the sky. So the highest south-west coordinate would be (11 1 9). And the lowest north-west coordinate is (0 0 0). Middle of environment/grid is (5 5 0). If I will start from the middle of environment/grid, but on the floor, face north and put one block, it coords will be (4 5 0). I can place blocks in this environment. For task: "Facing north, build three stacks of two orange blocks, then destroy the bottom orange block on all three stacks"; I will get a cords for built figure: ((2 4 0), (2 5 0), (2 6 0)). If I want a column/tower of 6 yellow blocks in the middle of the grid, the coords will be (5 5 0), (5 5 1), (5 5 2), (5 5 3), (5 5 4), (5 5 5). If I want a row with 3 blocks towards south: (5 5 0), (5 6 0), (5 7 0). What coords will be for tower of 6 yellow blocks in the middle of the grid? As answer, send me only list of coords.*

Next, we ask to add the color of the shape to the coordinate description, which should be chosen for a specific block.

*As answer, send me only list of cords. Add color digit to tuples array with the next color mapping: BLUE - 1 GREEN - 2 RED - 3 ORANGE - 4 PURPLE - 5 YELLOW - 6 and use 0 - for blocks needs to be removed*

Now, in the chatbot’s single-session mode, we sequentially present the instructions from the task without making any alterations to them.
C Instruction Augmentation Prompt

In this prompt, we also start by providing information about the environment like in B and the dialogues. Then, we ask to rewrite the dialogue as if the shapes were required to be built rotated by 180 degrees.

*I have a dialog with instructions describing how to build a 3D figure on a plane. The instructions have information on where to build the shapes and how to build them. I will consistently give you dialogues, and you have to rewrite the instructions so that they describe the construction of the same figure, only the location is rotated 180 degrees clockwise. Size of environment is (9, 11, 11) Remember: column and tower are vertical structures, line and row are horizontal. Do not change the instructions associated with the modification of vertical structures (keywords top and bottom). The output is only the final dialogue (be sure to save the ++ signs, I will then use them to separate the dialogue into instructions).*

D RL Approach Architecture

![Figure 4: The architecture of RL approach.](image)

E PPO Hyperparameters

<table>
<thead>
<tr>
<th>Hyperparameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adam learning rate</td>
<td>0.0001</td>
</tr>
<tr>
<td>$\gamma$ (discount factor)</td>
<td>0.99</td>
</tr>
<tr>
<td>Rollout</td>
<td>32</td>
</tr>
<tr>
<td>Clip ratio</td>
<td>0.1</td>
</tr>
<tr>
<td>Batch size</td>
<td>1024</td>
</tr>
<tr>
<td>Optimization epochs</td>
<td>1</td>
</tr>
<tr>
<td>Entropy coefficient</td>
<td>0.003</td>
</tr>
<tr>
<td>Value loss coefficient</td>
<td>0.5</td>
</tr>
<tr>
<td>GAE$_\lambda$</td>
<td>0.95</td>
</tr>
<tr>
<td>ResNet residual blocks</td>
<td>3</td>
</tr>
<tr>
<td>ResNet number of filters</td>
<td>64</td>
</tr>
<tr>
<td>LSTM hidden size</td>
<td>512</td>
</tr>
<tr>
<td>Activation function</td>
<td>ReLU</td>
</tr>
<tr>
<td>Network Initialization</td>
<td>orthogonal</td>
</tr>
<tr>
<td>Rollout workers</td>
<td>16</td>
</tr>
<tr>
<td>Environments per worker</td>
<td>32</td>
</tr>
<tr>
<td>Training steps</td>
<td>200000000</td>
</tr>
<tr>
<td>Random / IGLU structures mixing</td>
<td>30% / 70%</td>
</tr>
</tbody>
</table>