Retrieval-Augmented Code Generation for Situated Action Generation: A Case Study on Minecraft

Chalamalasetti Kranti[1], Sherzod Hakimov[1], David Schlangen[1,2]
[1]Computational Linguistics, Department of Linguistics
University of Potsdam, Germany
[2]German Research Center for Artificial Intelligence (DFKI), Berlin, Germany
{kranti.chalamalasetti, sherzod.hakimov, david.schlangen}@uni-potsdam.de

Abstract

In the Minecraft Collaborative Building Task, two players collaborate: an Architect (A) provides instructions to a Builder (B) to assemble a specified structure using 3D blocks. In this work, we investigate the use of large language models (LLMs) to predict the sequence of actions taken by the Builder. Leveraging LLMs' in-context learning abilities, we use few-shot prompting techniques, that significantly improve performance over baseline methods. Additionally, we present a detailed analysis of the gaps in performance for future work.

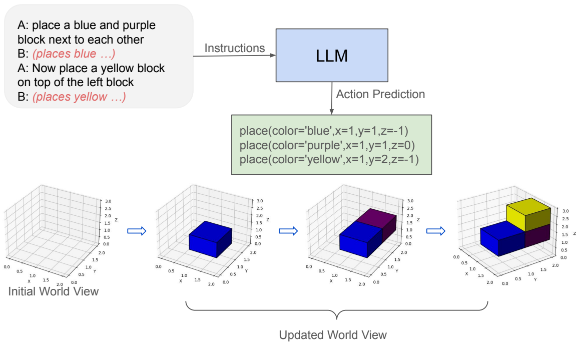

Figure 1: Illustration of the LLM interpreting block placement instructions. The initial world view is empty. The LLM receives instructions from User A and generates action predictions.

1 Introduction

Accurate action prediction is fundamental in developing interactive agents. These agents need to anticipate and respond effectively by interpreting the environment, including its contextual cues (Roy and Reiter, 2005; Goodrich and Schultz, 2007). A key part of this process is the interpretation of instructions and intentions in dialogues to predict appropriate subsequent actions (Chen and Mooney, 2011; Matuszek et al., 2012; Schlangen, 2023). As agents continuously predict actions based on the evolving context, precise action prediction becomes essential for ensuring seamless interaction and coordination (Winograd, 1971; Thomaz and Breazeal, 2008; Tellex et al., 2020; Ichter et al., 2022).

Building on these principles, our research focuses on modeling the Builder's sequence of actions (see Figure 1) in the Minecraft Collaborative Building Task (Narayan-Chen et al., 2019). This task requires close coordination between an Architect (A) and a Builder (B). The Architect provides instructions to the Builder, who assembles a specified structure using 3D blocks. This setup provides an ideal testbed for investigating how advanced computational models can interpret and predict actions based on natural language instructions.

Prior works (Jayannavar et al., 2020; Shi et al., 2022; Kiseleva et al., 2022) proposed end-to-end neural models for modeling builder actions prediction. However, achieving high accuracy for this task remains challenging. A key challenge lies in effectively utilizing richer dialogue history when modeling action sequence prediction. Meanwhile, LLMs have demonstrated impressive performance in various natural language processing tasks, primarily due to their in-context learning (Brown et al., 2020) capabilities. Few-shot prompting techniques, in particular, allow these models to generalize from a limited number of examples (Liu et al., 2023; Wei et al., 2022; Wu et al., 2023). This makes LLMs well-suited for tasks that require a nuanced understanding of longer contexts and action prediction (Xi et al., 2023; Liang et al., 2023; Singh et al., 2023; Driess et al., 2023; Vemprala et al., 2024).

In this work, we explore the application of LLMs to predict the Builder's actions. Taking advantage of their in-context learning capabilities, we prompted LLMs to predict these actions. To capture the nuances of this task, we adopted a retrieval-augmented approach (Lewis et al., 2020). This enhances the LLM's ability to predict action se-

Findings of the Association for Computational Linguistics: EMNLP 2024, pages 11159–11170
November 12-16, 2024 ©2024 Association for Computational Linguistics

quences that are not only syntactically correct but also contextually appropriate. Consequently, these predictions can be expressed either as textual descriptions or code-based commands. Leveraging the code-generation capabilities of LLMs (Chen et al., 2021), we frame the action prediction task as a code-generation problem. Specifically, we model the task as predicting a series of *pick()* and *place()* commands for the Builder.

While the proposed work focuses on a specific task within Minecraft, the underlying principles can be extended to more complex domains. Modeling action prediction across multiple dialogue turns as a code-generation task is well-suited for structured, repetitive workflows, such as those in industrial automation and robotics, where automation through code can streamline tasks that require precise coordination and context understanding.

2 Related Work

Minecraft Collaborative Building Task: Narayan-Chen et al. (2019) sourced human-to-human conversations for building structures in the Minecraft world. Building on this dataset (Jayannavar et al., 2020) modeled builder action sequences using an encoder-decoder architecture with dialogue history and world state representation. Further advancing this task (Shi et al., 2022) proposed a joint learning task for collaborative building and asking clarification questions. However, their approach focuses on generating action sequence of length *one*, rather than a series of actions. Despite these efforts, the task is far from being fully solved.

IGLU - Multi Turn Dataset: Mohanty et al. (2023) features a human collaborating with an AI agent to build target structures, using 31 out of 150 original Minecraft building tasks. Since we focus on action generation for human-human interactions, this dataset falls out of scope for our evaluation.

LLMs for Instruction Translation to a Code Snippet: Several research works (Chen et al., 2021; Huang et al., 2022; Zeng et al., 2023; Zhao et al., 2023) utilize LLMs for translating input natural language instructions to executable code snippets. These efforts span domains such as program synthesis (Wang et al., 2021; Touvron et al., 2023; Rozière et al., 2023; Hou and Ji, 2024), grounding dialogue (Chiu et al., 2023; Wu et al., 2024) and robot instruction generation (Liang et al., 2023; Singh et al., 2023; Driess et al., 2023; Vemprala et al., 2024; Kim et al., 2024). This capability of LLMs motivates us to formulate the builder action prediction task as a code generation task.

LLMs for Minecraft Collaborative Building Task Madge and Poesio (2024) use LLMs with text representation for action prediction, aligning closely with our proposal. However, we represent builder actions as code snippets and experiment on the Minecraft Dialogue dataset (Narayan-Chen et al., 2019), providing a detailed analysis of action prediction performance.

3 Dataset

Minecraft is used for studying collaborative tasks in a 3D voxel grid, with agents following architect's natural language instructions to build structures. The collaborative dialogue dataset (Narayan-Chen et al., 2019) has conversations for 150 target structures with varying levels of abstractions and complexity, grouped into 547 dialogue games, split into train (309 games, 3,792 turn-code pairs), development (101 games, 1,335 turn-code pairs) and test sets (137 games, 1,615 turn-code pairs). We used the *test set* for our evaluation.

3.1 Builder Action Transformation

Since the builder action prediction task is framed as a code-generation task, we convert the conversation from the format in the corpus (i.e., dialogue + action) into instruction + pseudo code. All the utterances of the builder and architect before each builder's action are aggregated into a single instruction. Builder actions involving *puts down* are converted to *place()* function, and actions involving *picks up* are converted to *pick()* function. The following is a sample representation of this conversion.[1]

INSTRUCTION: One block away from the edge, place a green block

BUILDER ACTION: Builder puts down a green block at X:0 Y:1 Z:4

CODE-REPRESENTATION: `place(color='green', x=0, y=1, z=4)`

INSTRUCTION: remove the middle block

BUILDER ACTION: Builder picks up a red block at X:0 Y:2 Z:0

CODE-REPRESENTATION: `pick(color='red', x=0, y=2, z=0)`

[1] The transformed pseudo code representations of Builder's actions are available at: https://github.com/clp-research/situatedactiongen-coderep

4 Experimental Setup

Task Formulation Formally, given a dialogue $\mathbf{d}$ with m turns $(\mathbf{x_1}, ..., \mathbf{x_m})$, the task is to generate the sequence of code snippets $(\mathbf{y}_1, ..., \mathbf{y}_m)$ that corresponds to each turn in a dialogue.

Few-shot Prompting Following standard prompting approaches (Brown et al., 2020; Liu et al., 2023; Wei et al., 2022; Wu et al., 2023), we adopt few-shot prompting to probe LLMs. Our prompt (see Figure 3 in Appendix) includes environment information such as the voxel grid size, available colors for the blocks and the quantity of each color. This is followed by dynamically adapted in-context examples suitable to the current turn instruction from the training set.

Ablation Study We first investigate how the building blocks of the prompt structure shown in Figure 3 (in Appendix) impact overall task performance. Using the validation set of the Minecraft Dialogue dataset for our ablation study, we observe that the prompt structure with all components is optimal for the action prediction task and the LLMs in our experiments, as demonstrated in Table 2. Specifically, omitting in-context examples results in poor performance, while excluding environment information reduced the score slightly. Consequently, we use the prompt featuring all components with three in-context examples to ensure the best performance.

Model Variants For this study, we utilized three advanced LLMs: GPT-4-o [2], Llama-3-8b [3], and Llama-3-70b [3]. These models are instruction-tuned, enabling them to generate code snippets based on input text instructions. GPT-4-o and Llama-3-70b were accessed via their respective APIs, incurring a cost of $12 for usage. For the Llama-3-8b model, experiments were conducted locally on a single A100 GPU with 80GB of memory, taking 10 hours in total. All models are queried with temperature 0 and `max_new_tokens` = 500.

We also experimented with CodeLlama-34b [4] model. However, the results were not satisfactory because the generated response contained new instructions generated by model (hallucinations) and it was impossible to evaluate the closeness of the prediction to the ground truth, and hence this model is excluded from the experimental results.

Retrieval-Augmented In-context Samples We use the pre-trained all-MiniLM-L6-v2 model from Sentence Transformers (Reimers and Gurevych, 2019) to compute the similarity between the current turn instruction and all turns in the training set. We use the cosine similarity to identify the top k most relevant examples. Based on the ablation study results (see Table 2 in Appendix), the top *three* turns and their corresponding builder actions are selected as in-context examples. The following is an example representation of this approach.

TEST INSTRUCTION: start with a column of 5 purple bricks
IN-CONTEXT EXAMPLES:
INSTRUCTION: add two lines of purple bricks
```
place(color='purple', x=0, y=1, z=4)
place(color='purple', x=0, y=1, z=5)
...
```
INSTRUCTION: place three yellow blocks in last row
```
place(color='yellow', x=4, y=1, z=0)
place(color='yellow', x=4, y=1, z=1)
...
```
INSTRUCTION: start with a column of 5 red bricks
```
place(color='red', x=-2, y=1, z=0)
place(color='red', x=-2, y=2, z=0) ...
```

LLM Fine Tuning In addition to few-shot prompting, we explored fine-tuning the Llama-3-8b model on the Minecraft building task to improve its generation of builder action sequences. We use the training set for the fine-tuning experiments. Additional details about the fine-tuning process are available in Appendix B.

Evaluation metrics We follow the same evaluation strategy as BAP (builder action prediction) baseline model (Jayannavar et al., 2020). The baseline model predicts action sequences in sentence form and, for evaluation purposes, converts these sequences into tuples of actions. Each tuple contains an action type (*pick* or *place*), color, and x, y, and z values. These tuples are generated for both the ground truth and the predicted action sequences. The net action tuples are then compared to compute the F1-Score. Similarly, in our approach, each dialogue turn prompts the LLM to generate action sequences (code) that provide information about a single block. These predicted sequences are compared against net ground truth by checking command type (*pick* or *place*), block color and X, Y, Z coordinates. This procedure is applied to all action sequences. We report the micro-averaged F1 on all dialogue turns.

[2] https://platform.openai.com/docs/models/gpt-4o
[3] https://ai.meta.com/blog/meta-llama-3/
[4] https://ai.meta.com/blog/code-llama-large-language-model-coding/

5 Results & Analysis

5.1 Overview

The results compared to the baseline Builder Action Prediction (BAP) model by Jayannavar et al. (2020), are presented in Table 1. GPT-4 achieved the best result (0.39) closely followed by Llama-3-70b (0.33). The fine-tuned version of Llama-3-8b showed a $\sim 6\%$ improvement over the vanilla version. Even though GPT-4 significantly outperforms the fine-tuned baseline, the upper bound for this task remains low. To understand this, we analysed the dialogues and identified references to spatial relations, real-world/geometric shapes, and anaphora. We then show how the models perform in these categories. Additionally, we identified two more factors complicating the interpretation of architect utterances, which may further impact action prediction performance.

5.2 Error Analysis

Spatial Prepositions: These are cases where the architect's utterances include prepositions that refer to a specific position in the grid, e.g. "put another blue block on *top* of it" or "two blocks on the ground, and one above on the *left*". Using Stanza (Qi et al., 2020) and spaCy [5] we extract phrases with the Part-of-Speech (POS) tags: adverb (ADP), noun (NN), and preposition (IN), then manually filter out non-spatial phrases resulting in 135 words. Examples include "left", "right", "top", "bottom", "down", "front", "back", "towards", "between", "behind", "opposite", "parallel", and "inverse". In the test set, 75.42% of the utterances include a spatial preposition, but GPT-4 correctly generates code for only 26.03% of them. Llama-3-70b closely follows by correctly predicting 22.66%, while Llama-3-8b achieves only 9.44%, highlighting significant challenges in interpreting spatial prepositions and a need for improvement.

Geometric and Real-World Shapes: Architect's utterances often include noun phrases referring to geometric shapes or real-world objects. Using the same procedure as above, we extracted 148 relevant nouns, including "trident", "chair", "pitchfork", "circle", and "rectangle". In the test set, 29.85% of utterances include one of these shapes, but GPT-4 correctly generated code for only 18.26% of them. Llama-3-70b generated 14.11% action sequences correctly, while Llama-3-8b managed to generate only 6.22% correctly. This indicates that while LLMs have extensive general knowledge, they struggle with code generation for shape references, necessitating further examination to address these challenges.

[5] https://spacy.io/

Model	F1
GPT-4	**0.39**
Llama-3-70b	0.33
Llama-3-8b	0.18
Llama-3-8b (fine-tuned)	0.19
BAP (fine-tuned)	0.21

Table 1: Micro-Average F1-score for the builder action prediction task. The BAP (fine-tuned) model results are reported for H2 in game history and with 4x data augmentation (Jayannavar et al., 2020)

Anaphora: Architect's utterances often include pronouns referring to previous concepts, e.g. "put another block next to *it*". Using the same procedure as above, we extracted and manually filtered these pronouns, identifying 16 words such as "that", "this", "those", "it". 46.81% of utterances in the test set include one of these pronouns but GPT-4 generated correct code only for 25.53% of them. Llama-3-70b showed an accuracy of 20.5%, while Llama-3-8b achieved an accuracy of 10.05%. Unlike spatial and geometric references, not all anaphora utterances indicate a reference to building concepts. They can also include acknowledgments or other types of information.

Builder Mistakes: These cases involve the builder making mistakes during the structure building process. These mistakes occur when the builder places a block and later removes it, leaving spurious action sequences—one *place* and one *pick*—in the ground truth action sequences. E.g."puts down a red block at X:-2 Y:0 Z: -1 followed by picks up a red block at X:-2 Y:0 Z:-1 (for a detailed example, please refer to Figure 5 in Appendix). Such cases cause evaluation mismatches. We filtered these by identifying *place* commands followed by *pick* commands at the same position and block color, finding that 23.3% of turns contained such mistakes. These inaccuracies in the ground truth lead to lower evaluation scores, as the model is penalized for errors that are not indicative of its true performance.

Underspecified Instructions: These are cases where the architect's utterance is underspecified, meaning it may lack specific details such as colour or precise location, or it may have multiple pos-

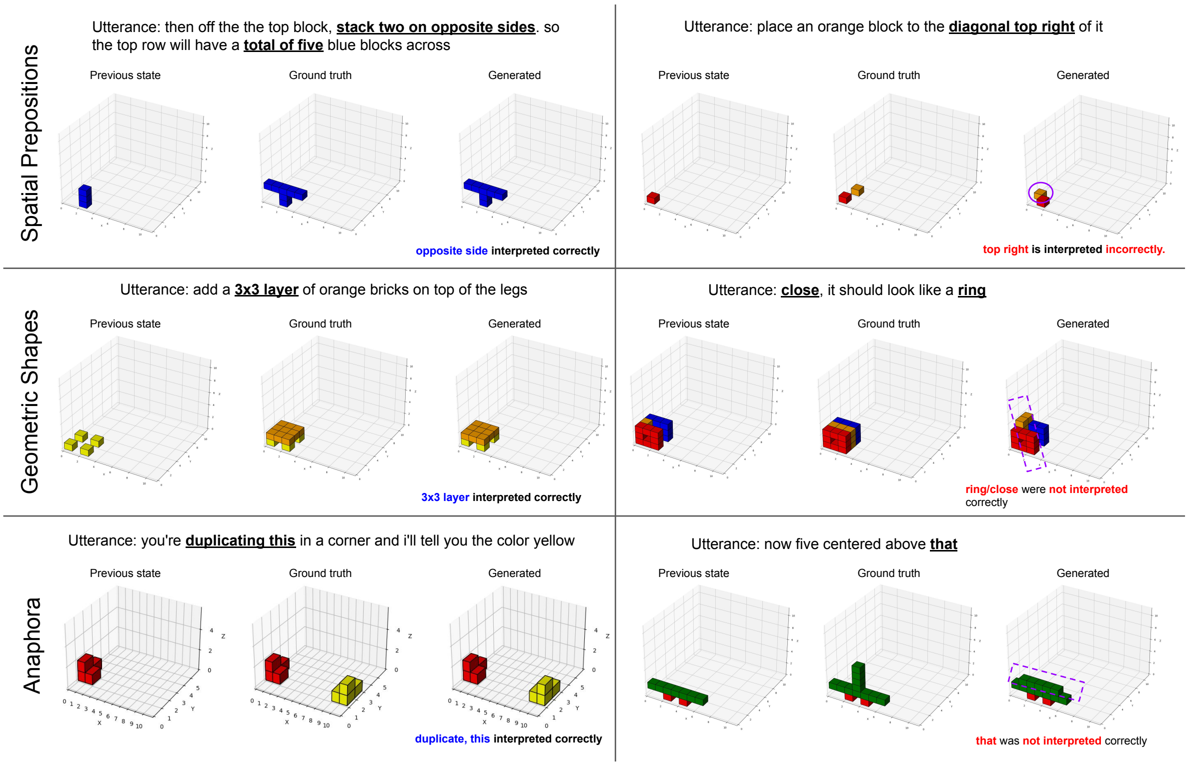

Figure 2: Voxel representations for sample turns that correspond to spatial preposition, geometric shape, and anaphora categories. Two samples for each category are given. Samples on the left side are generated correctly while samples on the right hand side have mistakes that are highlighted.

sible interpretations. To give an example, in the snippet shown in Figure 6 (in Appendix), there is no explicit indication in the dialogue history to interpret the instruction in a particular way, and the builder is left to decide on their own. Cases like this highlight the challenge of achieving a 100% match with the ground truth.

5.3 Qualitative Analysis

In Figure 2, we provide qualitative examples for GPT-4 that illustrate the scenarios for spatial prepositions, geometric shapes, and anaphora. We have included two samples for each scenario: one where the model response aligns with the ground truth (left), and another where it does not (right). Errors are marked by deviations in block positioning, color, or quantity, while correct interpretations align with the ground truth in both structure and placement. The model is able to handle spatial references such as "opposite" (e.g., stacking two blocks on opposite sides) and geometric shapes like "3x3". Additionally, the model correctly interprets repetitions such as "duplication", when the context is clear. However, it encounters challenges in accurately maintaining the exact count of blocks, translating words such as "close", "ring" into actions and resolving what "that" refers to. Overall the model struggles to handle tasks requiring more nuanced spatial reasoning, handling ambiguous references, and interpreting abstract geometric shapes.

6 Conclusion

We investigate prompting LLMs to build complex structures for the Minecraft building task. Predicting suitable builder action sequences for a given natural language instruction is challenging as these LLMs need to accurately interpret language abstractions, decode spatial co-references and reason about repetitions solely based on the in-context samples in the prompt. We compared multiple instruction-tuned LLMs, both closed and open-source. In addition to showcasing improved performance over baseline results, we also conduct an in-depth analysis of the generated responses across multiple dimensions. Although these models demonstrated a strong ability to accurately capture the action structure (*pick*, *place*), they struggled in dealing with spatial references, geometric shapes and anaphora. In the future, we plan to investigate model architectures that can address the shortcomings identified in the evaluated models.

Limitations

Like all other prompting approaches, our approach needs to be more robust to the usability of pre-trained large language models. (i) Not all grounded locations conform to the dimensions of the grid (ii) Fails in interpreting instructions involving agent's perspective such as "towards your right", "behind you", "away from you" etc. (iii) Struggles in understanding abstractions in the dialogue (iv) Since the study is carried out in a simulated world, extending it to real-world agents may lead to incorrect consequences..

Ethics Statement

The research uses an open-source dataset (Minecraft dialogue corpus) and open-access, closed-API based pre-trained large language models. While these models are accessible, their usage is subject to legal restrictions as outlined in their respective terms of service and licensing agreements. Minecraft dialogue corpus does not include players' personal, private information and does not contain any offensive conversations. However, the pre-trained LLMs, which inherit biases from their training data, may lead to code that favors certain styles and neglects others, hindering code portability. Another concern is the potential complexity of LLM-generated code, which can hinder end-user refinement and reuse. Moreover, it's crucial to ensure LLM-generated responses are free from harmful code, as their direct execution could impact the entire system.

Acknowledgements

The work reported here has been funded by the Bundesministerium für Bildung und Forschung (BMBF, German Federal Ministry of Research), project "COCOBOTS" (01IS21102A). We thank the anonymous reviewers for their helpful feedback.

References

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

David L. Chen and Raymond J. Mooney. 2011. Learning to interpret natural language navigation instructions from observations. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2011, San Francisco, California, USA, August 7-11, 2011*, pages 859–865. AAAI Press.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *CoRR*, abs/2107.03374.

Justin T. Chiu, Wenting Zhao, Derek Chen, Saujas Vaduguru, Alexander M. Rush, and Daniel Fried. 2023. Symbolic planning and code generation for grounded dialogue. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 7426–7436. Association for Computational Linguistics.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. Qlora: Efficient finetuning of quantized llms. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. 2023. Palm-e: An embodied multimodal language model. In *International Conference on Machine Learning, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 8469–8488. PMLR.

Michael A. Goodrich and Alan C. Schultz. 2007. Human-robot interaction: A survey. *Found. Trends Hum. Comput. Interact.*, 1(3):203–275.

Wenpin Hou and Zhicheng Ji. 2024. A systematic evaluation of large language models for generating programming code. *CoRR*, abs/2403.00894.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. 2022. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, volume 162 of *Proceedings of Machine Learning Research*, pages 9118–9147. PMLR.

Brian Ichter, Anthony Brohan, Yevgen Chebotar, Chelsea Finn, Karol Hausman, Alexander Herzog, Daniel Ho, Julian Ibarz, Alex Irpan, Eric Jang, Ryan Julian, Dmitry Kalashnikov, Sergey Levine, Yao Lu, Carolina Parada, Kanishka Rao, Pierre Sermanet, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Mengyuan Yan, Noah Brown, Michael Ahn, Omar Cortes, Nicolas Sievers, Clayton Tan, Sichun Xu, Diego Reyes, Jarek Rettinghouse, Jornell Quiambao, Peter Pastor, Linda Luu, Kuang-Huei Lee, Yuheng Kuang, Sally Jesmonth, Nikhil J. Joshi, Kyle Jeffrey, Rosario Jauregui Ruano, Jasmine Hsu, Keerthana Gopalakrishnan, Byron David, Andy Zeng, and Chuyuan Kelly Fu. 2022. Do as I can, not as I say: Grounding language in robotic affordances. In *Conference on Robot Learning, CoRL 2022, 14-18 December 2022, Auckland, New Zealand*, volume 205 of *Proceedings of Machine Learning Research*, pages 287–318. PMLR.

Prashant Jayannavar, Anjali Narayan-Chen, and Julia Hockenmaier. 2020. Learning to execute instructions in a minecraft dialogue. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 2589–2602. Association for Computational Linguistics.

Yeseung Kim, Dohyun Kim, Jieun Choi, Jisang Park, Nayoung Oh, and Daehyung Park. 2024. A survey on integration of large language models with intelligent robots. *CoRR*, abs/2404.09228.

Julia Kiseleva, Ziming Li, Mohammad Aliannejadi, Shrestha Mohanty, Maartje ter Hoeve, Mikhail Burtsev, Alexey Skrynnik, Artem Zholus, Aleksandr I. Panov, Kavya Srinet, Arthur Szlam, Yuxuan Sun, Marc-Alexandre Côté, Katja Hofmann, Ahmed Awadallah, Linar Abdrazakov, Igor Churin, Putra Manggala, Kata Naszádi, Michiel van der Meer, and Taewoon Kim. 2022. Interactive grounded language understanding in a collaborative environment: IGLU 2021. *CoRR*, abs/2205.02388.

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. 2023. Code as policies: Language model programs for embodied control. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*, pages 9493–9500. IEEE.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Comput. Surv.*, 55(9):195:1–195:35.

Chris Madge and Massimo Poesio. 2024. Large language models as minecraft agents. *CoRR*, abs/2402.08392.

Cynthia Matuszek, Evan Herbst, Luke Zettlemoyer, and Dieter Fox. 2012. Learning to parse natural language commands to a robot control system. In *Experimental Robotics - The 13th International Symposium on Experimental Robotics, ISER 2012, June 18-21, 2012, Québec City, Canada*, volume 88 of *Springer Tracts in Advanced Robotics*, pages 403–415. Springer.

Shrestha Mohanty, Negar Arabzadeh, Julia Kiseleva, Artem Zholus, Milagro Teruel, Ahmed Awadallah, Yuxuan Sun, Kavya Srinet, and Arthur Szlam. 2023. Transforming human-centered AI collaboration: Redefining embodied agents capabilities through interactive grounded language instructions. *CoRR*, abs/2305.10783.

Anjali Narayan-Chen, Prashant Jayannavar, and Julia Hockenmaier. 2019. Collaborative dialogue in minecraft. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 5405–5415. Association for Computational Linguistics.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D. Manning. 2020. Stanza: A python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, ACL 2020, Online, July 5-10, 2020*, pages 101–108. Association for Computational Linguistics.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. 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 2019, Hong Kong, China, November 3-7, 2019*, pages 3980–3990. Association for Computational Linguistics.

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

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code llama: Open foundation models for code. *CoRR*, abs/2308.12950.

David Schlangen. 2023. On general language understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, pages 8818–8825. Association for Computational Linguistics.

Zhengxiang Shi, Yue Feng, and Aldo Lipani. 2022. Learning to execute actions or ask clarification questions. In *Findings of the Association for Computational Linguistics: NAACL 2022, Seattle, WA, United States, July 10-15, 2022*, pages 2060–2070. Association for Computational Linguistics.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: program generation for situated robot task planning using large language models. *Auton. Robots*, 47(8):999–1012.

Chan Hee Song, Brian M. Sadler, Jiaman Wu, Wei-Lun Chao, Clayton Washington, and Yu Su. 2023. Llm-planner: Few-shot grounded planning for embodied agents with large language models. In *IEEE/CVF International Conference on Computer Vision, ICCV 2023, Paris, France, October 1-6, 2023*, pages 2986–2997. IEEE.

Stefanie Tellex, Nakul Gopalan, Hadas Kress-Gazit, and Cynthia Matuszek. 2020. Robots that use language. *Annu. Rev. Control. Robotics Auton. Syst.*, 3:25–55.

Andrea Lockerd Thomaz and Cynthia Breazeal. 2008. Teachable robots: Understanding human teaching behavior to build more effective robot learners. *Artif. Intell.*, 172(6-7):716–737.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. Llama: Open and efficient foundation language models. *CoRR*, abs/2302.13971.

Sai Vemprala, Rogerio Bonatti, Arthur Bucker, and Ashish Kapoor. 2024. Chatgpt for robotics: Design principles and model abilities. *IEEE Access*, 12:55682–55696.

Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 8696–8708. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Terry Winograd. 1971. Procedures as a representation for data in a computer program for understanding natural language.

Guande Wu, Chen Zhao, Cláudio T. Silva, and He He. 2024. Your co-workers matter: Evaluating collaborative capabilities of language models in blocks world. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, pages 4941–4957. Association for Computational Linguistics.

Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas A. Funkhouser. 2023. Tidybot: personalized robot assistance with large language models. *Auton. Robots*, 47(8):1087–1102.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, Rui Zheng, Xiaoran Fan, Xiao Wang, Limao Xiong, Yuhao Zhou, Weiran Wang, Changhao Jiang, Yicheng Zou, Xiangyang Liu, Zhangyue Yin, Shihan Dou, Rongxiang Weng, Wensen Cheng, Qi Zhang, Wenjuan Qin, Yongyan Zheng, Xipeng Qiu, Xuanjing Huan, and Tao Gui. 2023. The rise and potential of large language model based agents: A survey. *CoRR*, abs/2309.07864.

Andy Zeng, Maria Attarian, Brian Ichter, Krzysztof Marcin Choromanski, Adrian Wong, Stefan Welker, Federico Tombari, Aveek Purohit, Michael S. Ryoo, Vikas Sindhwani, Johnny Lee, Vincent Vanhoucke, and Pete Florence. 2023. Socratic models: Composing zero-shot multimodal reasoning with language. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. 2023. A survey of large language models. *CoRR*, abs/2303.18223.

A Prompt Structure

The proposed task aims to investigate the capability of LLMs in accurately predicting actions based on input instructions. To generate an accurate action sequence, the models must understand the environment, follow dialogue history, and interpret the input instruction within the current context.

To achieve this, we construct a multi-part prompt, as illustrated in Figure 3, to query the LLM. This prompt includes several components: system information, environment information, task information, context information, and other guidelines. The system information specifies the expected behavior for the LLM. The environment information provides detailed information about the build region, its boundaries, and block properties. Specifically, the Minecraft dialogue corpus limits the build region to an 11x9x11 grid, and the builder is provided with an inventory of 120 blocks in six colors $colour \in (red, blue, orange, purple, yellow, green)$. The task information outlines the format of the expected outcome.

Following this, the context information includes in-context examples relevant to the current test instruction. Adopting the approach recommended by Song et al. (2023), we use adaptive prompts, dynamically retrieving examples from the training set that are similar to the current test instruction. Figure 4 shows how the in-context examples change based on the test instruction. This structure offers detailed information for the LLM to generate action sequences for the input instructions within the given environment.

B Fine tuning

Since the out-of-the-box performance of the LLMs sets a lower bound for the action prediction task, we explore fine-tuning to see if it improves task performance. We use the training set of the Minecraft Dialogue corpus and fine-tune the Llama-3-8b model. This deliberate choice is made because it is the smallest open-source model we have used and has the lowest performance among all models. For the fine-tuning process, we use Q-LORA (Dettmers et al., 2023) to reduce the memory footprint during fine-tuning by applying low-rank adaptations, allowing efficient fine-tuning on limited hardware.

We experimented with hyperparameter changes (number of epochs, learning rate) and finally chose the optimal performance setting, which includes 15 epochs and a learning rate of 0.0002. The model is configured using the Adam optimizer with a batch size of 32. The training process spans 15 epochs, with an early stopping condition based on evaluation loss to prevent over-fitting. Specifically, if the validation loss does not decrease for 5 consecutive epochs, the training is stopped. The validation set is used for evaluation during training to monitor the model's performance and guide the early stopping mechanism. We experimented with hyperparameter changes (number of epochs, learning rate) and finally chose the optimal performance setting, which includes 15 epochs and a learning rate of 0.0002. The model is configured using the Adam optimizer with a batch size of 32. The training process spans 15 epochs, with an early stopping condition based on evaluation loss to prevent overfitting. Specifically, if the validation loss does not decrease for 5 consecutive epochs, the training is stopped. The validation set is used for evaluation during training to monitor the model's performance and guide the early stopping mechanism.

This fine-tuned model is then used for testing on the test set and is indicated as the Llama-3-8b (fine-tuned) model in Table 1. Compared to the baseline (pre-trained) model, the fine-tuned model shows an enhancement of $\sim 6\%$ in the F1-score. This improvement, although marginal, indicates a potential to improve the model's ability for the action prediction task.

TEMPLATE A.0.1

System Info

You are an expert at interpreting natural language instructions and translating them into specific actions for placing colored blocks within a 3D grid.

Environment Info

The environment is structured as an 11x9x11 grid, with each cell representing a single block. The grid is organized such that columns align along the X-axis and rows along the Z-axis, employing Python indexing for cell identification. The X and Z axes range from -5 to 5, encompassing length and width, while the Y-axis ascends from 1 to 9, indicating height. This grid, following an XZY orientation, places X and Z as horizontal dimensions and Y as the vertical dimension. The ground level is defined at Y=1, with higher Y-values signifying increased elevation, thereby representing the vertical aspect of the environment. There are 20 blocks for each of the following colors: blue, orange, red, green, yellow, and purple.

Please import the following functions for object interactions:
pick(color,x,y,z): This function picks the object at the specified location.
place(color,x,y,z): This function places the object at the specified location.

You should only use these specified functions and not create any hypothetical functions.

Please use the pick function only to select the pieces that you have previously placed on the board. You can safely assume that the available blocks are automatically selected, and therefore, you can directly use the "place()" function without needing to call the "pick()" function first.

Task Info

Your task is to interpret natural language instructions and generate the corresponding code representation. For each instruction labeled "Instruction" please respond with code under the label "Output" followed by a newline.

Context Info

$INCONTEXT_SAMPLES

Other Info

Do not generate any other text/explanations. Use python code to express the solution. Please keep the solutions simple and clear. Do not use loops, comments in the solution. Do not generate the instructions on your own. Stick to the given format. Think step by step.

Let's get started.

$TEST_INSTRUCTION

Figure 3: Prompt template used for the action prediction task. The system information specifies system level behavior, the environment information indicates the environment details of the user-agent environment, the context information describes the in-context examples, task information indicates the specific response format to follow.

TEMPLATE A.0.2

INCONTEXT_SAMPLES

The INCONTEXT_SAMPLES varies for each turn. Below is an example

Instruction

Mission has started.. This structure looks fun- nothing will touch the ground!. cool!. We'll start with a row of 4 purple bricks, a layer above the ground

Output

```
place(color='purple',x=1,y=1,z=1)
place(color='purple',x=1,y=2,z=1)
pick(color='purple',x=1,y=1,z=1)
place(color='purple',x=0,y=2,z=1)
place(color='purple',x=-1,y=2,z=1)
place(color='purple',x=-2,y=2,z=1)
```

Instruction

Mission has started.. we'll start with two lines of purple bricks, parallel to one another. seperated by an empty space

Output

```
place(color='purple',x=1,y=1,z=-2)
pick(color='purple',x=1,y=1,z=-2)
```

Instruction

Mission has started.. We'll start with a column of 5 red bricks

Output

```
place(color='red',x=0,y=1,z=1)
place(color='red',x=0,y=2,z=1)
place(color='red',x=0,y=3,z=1)
place(color='red',x=0,y=4,z=1)
place(color='red',x=0,y=5,z=1)
```

TEST_INSTRUCTION

Mission has started.. hi again!. Hello! This structure will start with a column of 5 purple bricks. It's all purple.

Figure 4: Retrieval of relevant in-context examples based on current test instruction

Prompt	F1
System Info + Env Info + Task Info + Context Info (Zero Samples) + Other Info	0.15
System Info + Env Info + Task Info + Context Info (One Sample) + Other Info	0.17
System Info + Env Info + Task Info + Context Info (Two Samples) + Other Info	0.18
System Info + Env Info + Task Info + Context Info (Three Samples) + Other Info	0.18
System Info + Env Info + Task Info + Context Info (Four Samples) + Other Info	0.18
System Info + Env Info + Task Info + Context Info (Five Samples) + Other Info	0.18
Env Info + Task Info + Context Info (Three Samples) + Other Info	0.19
System Info + Task Info + Context Info (Three Samples) + Other Info	0.17
System Info + Env Info + Context Info (Three Samples) + Other Info	0.17
System Info + Env Info + Context Info (Three Samples)	0.17

Table 2: Ablation study for the number of in-context examples and components of the prompt structre on validation split of the Minecraft dataset using Llama-3-8b.

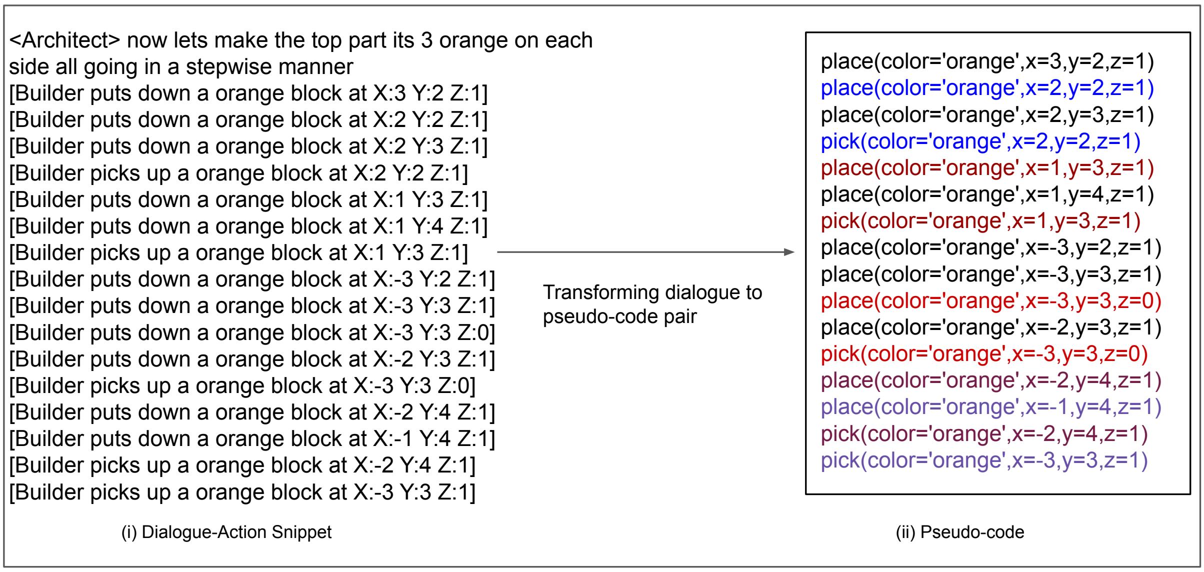

Figure 5: Excerpt of an utterance that contains the builder mistakes from the game-id: B29-A1-C151-1524078449685. The action sequence pairs where an item is first placed and later picked up is highlighted with the same colour.

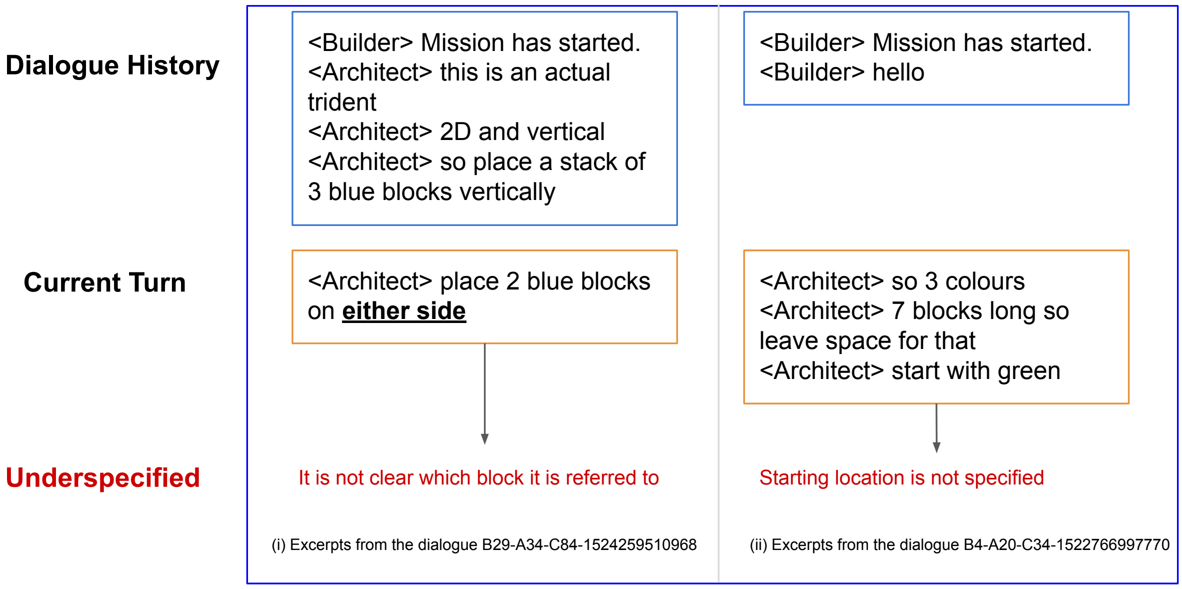

Figure 6: Examples utterances from the architect where the given command is underspecified