From Templates to Natural Language: Generalization Challenges in Instruction-Tuned LLMs for Spatial Reasoning

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

Instruction-tuned large language models (LLMs) have shown strong performance on a variety of tasks; however, generalizing from synthetic to human-authored instructions in grounded environments remains a challenge for them. In this work, we study generalization challenges in spatial grounding tasks where models interpret and translate instructions for building object arrangements on a 2.5D grid. We fine-tune LLMs using only synthetic instructions and evaluate their performance on a benchmark dataset containing both synthetic and human-authored instructions. Our results reveal that while models generalize well on simple tasks, their performance degrades significantly on more complex tasks. We present a detailed error analysis of the gaps in instruction generalization.

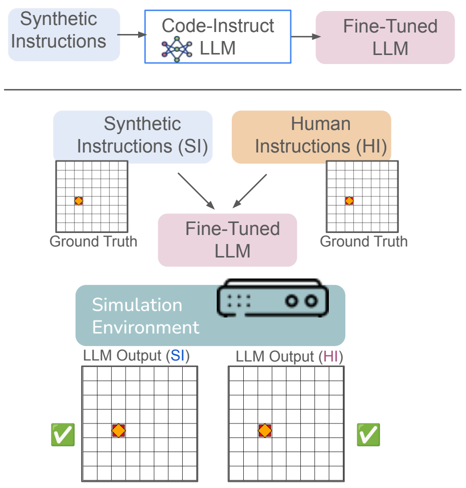


Figure 1: Fine-tuned LLMs trained on synthetic instructions are evaluated on both synthetic (SI) and human (HI) instructions. A virtual simulator verifies whether generated response matches the gold configuration.

1 Introduction

Accurate spatial grounding (Green et al., 2006; Brenner, 2007) is important for effective human-robot interaction (Bisk et al., 2016; Shridhar and Hsu, 2018; Hatori et al., 2018). It involves the robot interpreting spatial references (Hüttenrauch et al., 2006) and relational cues expressed in natural language instructions (Dang-Vu et al., 2015; Paul et al., 2018; Tellex et al., 2020; Bisk et al., 2020). Large Language Models (LLMs) are increasingly used in robotics pipelines (Ichter et al., 2022; Wang et al., 2024b; Lim et al., 2024; Macaluso et al., 2024) to translate such instructions into high-level plans and then into low-level robot actions. While these models have shown progress in instruction-following tasks, they continue to face challenges (Liu et al., 2023; Ding et al., 2023; Jiang et al., 2025) when grounding spatial language references.

Human instructions are often abstract (Minsky, 1980), underspecified (Hendriks et al., 2014), and context-dependent. For example, a user might say *"put the red block on top of that blue one"* in one case, and *"make a staircase"* in another. These variations rely on shared context and conceptual inference, making them harder to interpret (Kranti et al., 2024a; Chaturvedi et al., 2024) than synthetic or templated instructions, which are designed to be direct and explicit.

To explore these issues in a controlled setting, we investigate them in the context of a structure-building task, where instructions guide the manipulation of objects on a grid. We leverage an existing dataset (Kranti et al., 2024b) that includes both synthetic and human-authored instructions for the same grounded spatial reasoning task. Each task involves building specific arrangements of colored shapes on an 2.5D grid with size 8×8, referred to as *"boards."* Each board is paired with both synthetic and human-authored instructions, enabling direct comparison of model behavior (see Figure 1) across instruction styles within a shared task space.

We evaluate how well pre-trained LLMs perform with both synthetic and human-authored instructions across different board types. Furthermore, we fine-tune instruction-tuned LLMs using only the synthetic instructions and evaluate them whether they generalize to human-authored ones. Our results show that while fine-tuning improves model performance, and models achieve high execution scores on simple boards, performance gains are limited on regular boards. This suggests that instruction generalization is closely tied to the compositional and structural complexity of the task, and that fine-tuning solely on synthetic data is insufficient for robust transfer to instructions that require more abstract or relational understanding.

Our contributions are as follows: (a) we evaluate pre-trained and fine-tuned LLMs on both synthetic and human-authored instructions, and report quantitative performance across different board types (simple vs. regular); (b) we conduct a detailed qualitative analysis of model responses before and after fine-tuning, categorizing the types of errors across instruction styles and board complexity; (c) we analyze the embedding similarity of human-authored instructions, to their synthetic counterparts, the number of shapes involved, and how these factors correlate with model performance; and (d) we test the generalization of fine-tuned models on two related, out-of-domain code generation tasks, and report the performance related to the generalization.

2 Related Work

Spatial Grounding with LLMs Spatial grounding refers to the task of interpreting spatial references in natural language and linking them to a target environment. Recent work has explored the use of LLMs for tasks such as block manipulation (Valmeekam et al., 2023; Goldberg et al., 2024), table setting (Liang et al., 2023; Vemprala et al., 2024), tool usage (Xu et al., 2023) and collaborative assembly (Lim et al., 2024; Macaluso et al., 2024; Joglekar et al., 2024). Our work is inspired by these efforts but differs in how the grounding is performed. Prior approaches generate sequences of atomic actions or low-level code that are directly executable in the environment. In contrast, we focus on generating abstract functions that encode spatial semantics but are not immediately grounded; these functions are interpreted and executed by a downstream system. This abstraction allows us to analyze how well LLMs encode spatial structure independently of immediate action execution.

Instruction-to-Code Translation for Spatial Tasks Instruction-to-code translation focuses on converting natural language inputs into executable programs or structured representations (Singh et al., 2023; Huang et al., 2023a,b; Wang et al., 2024a; Hu et al., 2024). Our work extends these efforts by comparing model performance across two instruction styles, synthetic and human-authored, for the same spatial task. This setup enables us to study generalization and instruction-following behavior under consistent task conditions.

Generalization from Synthetic to Human Instruction Instruction-tuned models often struggle to generalize well from synthetic to natural language (Li et al., 2023; Nwankwo et al., 2025; Shi et al., 2025; Nadas et al., 2025). This reflects the challenges of handling variation in human-authored instructions. We extend this to a spatial grounding task and report similar failures in generalization despite high performance on synthetic tasks.

Fine-tuning on Synthetic vs. Human Instructions Prior work has explored fine-tuning LLMs for collaborative structure building in environments like MINECRAFT (Kranti et al., 2024a; Chaturvedi et al., 2024). These studies typically fine-tune on a mix of synthetic and human-authored instructions and report limited generalization. Furthermore, MINECRAFT dataset (Narayan-Chen et al., 2019) does not distinguish based on complexity and structures with and without object repetitions, making it difficult to isolate the impact of fine-tuning. Our work builds on these by analyzing performance gaps across instruction styles and board types that affect generalization.

3 Task Formulation and Datasets

Task: it is framed as a two-player game between a programmer and a cobot (collaborative robot), where there is no intermediate feedback and execution happens only at the end of the interaction. The programmer (instruction giver) is given a target board and instructs the cobot (instruction follower) to translate the instruction into code. The code generated by the cobot is executed in a virtual simulation environment and evaluated for correctness.

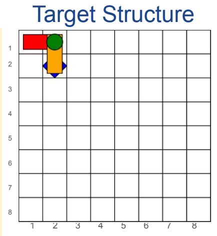
Boards: We use the SARTCo (Kranti et al., 2024b) dataset for finetuning and the main testbed for our experiments. It consists of 260 human-authored instructions for a structure building task

Code Representation

```
def bwbs(board, colors, x, y):
  shapes = ['bridge-h', 'washer',
            'bridge-v', 'screw']
  for shape, color, dx, dy in zip(shapes,
          colors, [0, 1, 0, 0], [0, 1, 1, 1]):
    put(board, shape, color, x + dx, y + dy)

bwbs(board, ('red', 'blue', 'yellow','green'),
         x=0, y=0)
```

Target Structure



Synthetic Instruction

Place a red bridge horizontally in the 1st row, 1st column. Place a blue washer in the 2nd row, 2nd column. Place a yellow bridge vertically in the 1st row, 2nd column. Place a green screw in the 1st row, 2nd column.

Human-Authored Instruction

Place a red horizontal bridge in the 1st row, 1st column. Add a blue washer below the right side of the bridge. Stack a yellow vertical bridge on top of the right side of the birdge and the washer. Stack a blue screw on top of the upper part of the yellow bridge.

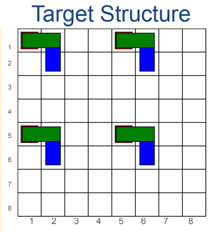
Figure 2: A simple board example showing how code (left) maps to a target structure (right), with both synthetic and human-authored instructions describing the same configuration. Models are expected to generate both the function definition and its usage based on the given instruction.

Code Representation

```
def nbb(board, colors, x, y):
  shapes = ['nut', 'bridge-v', 'bridge-h']
  for shape, color, dx, dy in zip(shapes,
          colors, [0, 0, 0], [0, 1, 0]):
    put(board, shape, color, x + dx, y + dy)

for row in [0, 4]:
  for col in [0, 4]:
    nbb(board, colors=['red', 'blue',
          'green'], x=row, y=col)
```

Target Structure



Synthetic Instruction

Place a 'nbb' object in the first, and fifth columns of the first row. Then, repeat this placement pattern in the fifth row. Use only these colors: ['red', 'blue', 'green'] for the 'nbb' object.

Human-Authored Instruction

Put a nbb object in the top left corner. The nut is red and the horizontal bridge on top is green. The vertical bridge is blue. Place another nbb object in the same colors in the same row, in column 5. Add two more nbb objects in the same colors in row 5, in the same two columns as the first two objects.

Figure 3: A regular board example illustrating repeated structural patterns defined by code (left), their corresponding target structure (right), and matching synthetic and human-authored instructions. The function definition is provided in the prompt; models are expected to generate only the usage code (e.g., nested for loops) based on the instruction.

on a 2.5D [1] grid. Each instance in the dataset includes a triplet: *a code representation*, *a target board*, and *a natural language instruction*. An example consisting of the target board, an abstract Python function, and two styles of instructions, is shown in Figure 2 and Figure 3. Each target board is a specific arrangement of objects on a grid. An *object* is a composition of shapes such as *washers*, *screws*, *nuts*, and *bridges*. Boards are labeled as either *simple*, with non-repetitive arrangements, or *regular*, with repetitive arrangements. The goal is to generate this semantically grounded function from the instruction. This setup reflects instruction-following scenarios and enables evaluation of how well LLMs can spatially ground such instructions.

Instructions: they are of two types: synthetic & Human-authored. Synthetic instructions are generated using a template-based grammar. They are structurally consistent and semantically explicit. They are available in both single-turn and multi-turn formats. Human-authored instructions are free-form language inputs for the same target structure. They are available only in single-turn format. Both types are aligned to the same structure building task.

Learning on Synthetic Instructions: we prompt LLMs with both instruction types and evaluate the performance by executing the generated code in a virtual simulation. The output is compared to the target to compute execution success, defined as the proportion of cases where the model output matches the target board. We fine-tune models on synthetic instructions and evaluate them on both synthetic and human-authored instructions. This setup tests how well models generalize from synthetic to human language in the same task domain.

Cross-domain Datasets: We also evaluate cross-domain generalization using the HEXAGONS (Lachmy et al., 2022) and TIDYBOT (Wu et al., 2023) datasets. These datasets help assessing whether fine-tuned models are able to perform on different but related spatial instruction tasks.

4 Experimental Setup

We evaluate a diverse set of LLMs with varying sizes and architectures, focusing on their ability to interpret grounded spatial instructions and generate executable action sequences. These models are used as-is (few-shot) and fine-tuned on synthetic data only, without exposure to human-authored instructions during training. We use the *clembench* (Chalamalasetti et al., 2023) framework to manage player interactions.

4.1 Evaluated Models & Parameters

Models: We include both code-centric models (QWEN2.5-CODER-7B, 32B) and general-purpose, instruction-following models (QWEN3-8B, 32B; LLAMA-3.1-8B, 3.3-70B). All models are loaded in 4-bit quantized precision using the UNSLOTH li-

[1]A 2D grid with stacking support in each cell and without the complexity of full 3D simulation.

brary[2]. We compared these models against the commercial ones: GPT-4o and CLAUDE-4-SONNET.

Fine-Tuning Details: Models are fine-tuned on synthetic instruction–code pairs, where the input is a synthetic natural language instruction and the output is the corresponding Python code representation of the board. The training set (Kranti et al., 2024b) comprises a mixture of simple boards (1072 boards) and regular boards (1168 boards). We adopt a chat-style prompt format, including context and environment details (more details in Appendix A.1), followed by the instruction and target code. We use the QLoRA configuration provided by UNSLOTH with the following hyperparameters: 3 *epochs*, 20 *steps*, `adam-8bit` optimizer, rank $r = 16$, `lora_alpha=16`, `lora_dropout=0.10`, batch size=8, and learning rate=1e$-$4.

Hyperparameter Selection and Ablations: To determine optimal fine-tuning settings, we conducted a comprehensive ablation study varying data composition, ordering, and hyperparameters. We evaluated models trained on (i) only simple boards, (ii) only regular boards, and (iii) combined boards with/without shuffling. Additionally, we explored the impact of training sample sizes (100-to-1000 per board type), batch sizes (2-to-8), learning rates (1e$-$4, 2e$-$4), epoch counts (1-to-5), `lora_dropout` (0.10, 0.15, 0.20) and early stopping patience(2, 3). These studies informed our final configuration: combined dataset with shuffled samples, full training set, batch size 8, 3 epochs, learning rate 1e$-$4, and dropout 0.10. More details and ablation study results are available in Appendix A.3.1.

4.2 Evaluation Metrics

We assess model performance on instruction-following capability, execution accuracy, and cross-domain robustness.

Error Rate: A metric capturing instruction adherence based on predefined response constraints (output formatting) in the prompt (Appendix A.1). In the clembench setup, responses that violate these constraints are marked as `abort`.

Execution Success Rate: Our primary task success metric. Each model-generated function is executed in a virtual simulator and compared against the target board (an 8×8 grid with shape type, color, position, and stacking order). A prediction is successful if the resulting board configuration exactly matches the ground truth in component type, color, spatial placement, and stacking sequence.

Human Baseline: To contextualize model performance, we include a human-generated baseline. We developed a simple user interface (see Figure 10 in Appendix) that shows the human-authored instructions (by a different annotator) on the left and the setup to reconstruct the target board on the right. We hired an annotator and asked to reconstruct target boards. These outputs were executed using the same virtual simulator and computed the same *Execution Success Rate* metric, allowing direct comparison between model and human performance.

Cross-domain robustness: We measure generalization by comparing model performance before and after fine-tuning. Specifically, we evaluate whether models trained on synthetic instructions generalize to (1) human-authored instructions in the same task domain (structure building: synthetic $\rightarrow$ human), and (2) structurally similar but domain-shifted tasks such as drawing hexagons (HEXAGONS) and sorting objects (TIDYBOT).

5 Results

We present quantitative and qualitative results, followed by cross-domain generalization analysis. Quantitative results are organized first by board type (simple and regular) for human-authored instructions, and then by instruction type (synthetic and human-authored) for overall comparison.

5.1 Quantitative Analysis

Performance based on Board Type Table 1 presents model accuracy on simple and regular boards for human-authored instructions. Most models perform better on simple boards. `Llama3.3-70B` and `Qwen2.5-Coder-32B` achieve higher scores of 0.69 and 0.68, respectively. While the scores also improve for regular boards, the gains are smaller. `Qwen2.5-Coder-32B` achieves 0.54 accuracy. `LLaMA3.1-8B` and `Qwen3-8B` have lower scores, primarily due to hallucinations or failure to follow the expected response format, resulting in aborted executions (see abort rate in Table 1). Although fine-tuning improves accuracy, the gap with the human upper bound remains substantial.

Smaller models (`Llama3.1-8B`, `Qwen3-8B`) perform better on regular boards than on simple boards. This may be due to differences in the expected output structure. For regular boards, the target code

[2] https://unsloth.ai/

Model	Abort Rate ↓				Performance ↑			
	SB		RB		SB		RB	
	Before	After	Before	After	Before	After	Before	After
Qwen2.5-Coder-7B	0.00	0.00	1.00	0.00	0.07	0.40	0.07	0.35
Llama3.1-8B	0.00	0.11	0.00	0.09	0.00	0.08	0.00	0.21
Qwen3-8B	1.00	0.38	0.95	0.59	0.00	0.09	0.00	0.12
Qwen2.5-Coder-32B	0.00	0.00	0.00	0.00	0.14	**0.68**	0.23	**0.54**
Qwen3-32B	0.95	0.00	0.97	0.00	0.00	0.53	0.00	0.38
Llama3.3-70B	0.00	0.00	0.72	0.06	0.11	**0.69**	0.07	0.33
GPT-4o	0.00	–	0.00	–	0.72	–	0.55	–
Claude-4(sonnet)	0.00	–	0.00	–	0.88	–	0.60	–
Human-Baseline	–	–	–	–	**0.98**		**0.76**	

Table 1: Abort rate and task performance (accuracy) on simple boards (SB) and regular boards (RB) from human-authored set, before and after fine-tuning on the synthetic set. While some models show gains on SB, improvements on RB are low.

Model	FSG		FSC	
	SB	RB	SB	RB
Qwen2.5-Coder-7B	0.25	0.25	0.23	0.19
Llama3.1-8B	0.02	0.28	0.11	0.29
Qwen3-8B	0.05	0.10	0.05	0.29
Qwen2.5-Coder-32B	0.45	**0.53**	0.32	**0.51**
Qwen3-32B	0.35	0.24	0.42	0.54
Llama3.3-70B	**0.54**	0.32	**0.55**	0.15

Table 2: Model performance with RB prompt styles (FSG: Function Signature, FSC: Function Schematic), evaluated on SB (simple) and RB (regular) boards.

Model	FSG		FSC	
	SB	RB	SB	RB
Qwen2.5-Coder-7B	0.00	0.00	0.23	0.19
Llama3.1-8B	0.01	0.00	0.31	0.00
Qwen3-8B	0.65	0.51	0.55	0.02
Qwen2.5-Coder-32B	0.00	0.00	0.00	0.00
Qwen3-32B	0.00	0.00	0.01	0.00
Llama3.3-70B	0.00	0.00	0.00	0.52

Table 3: Effect of RB prompt styles (FSG: Function Signature, FSC: Function Schematic) on abort rates for SB (simple) and RB (regular) boards.

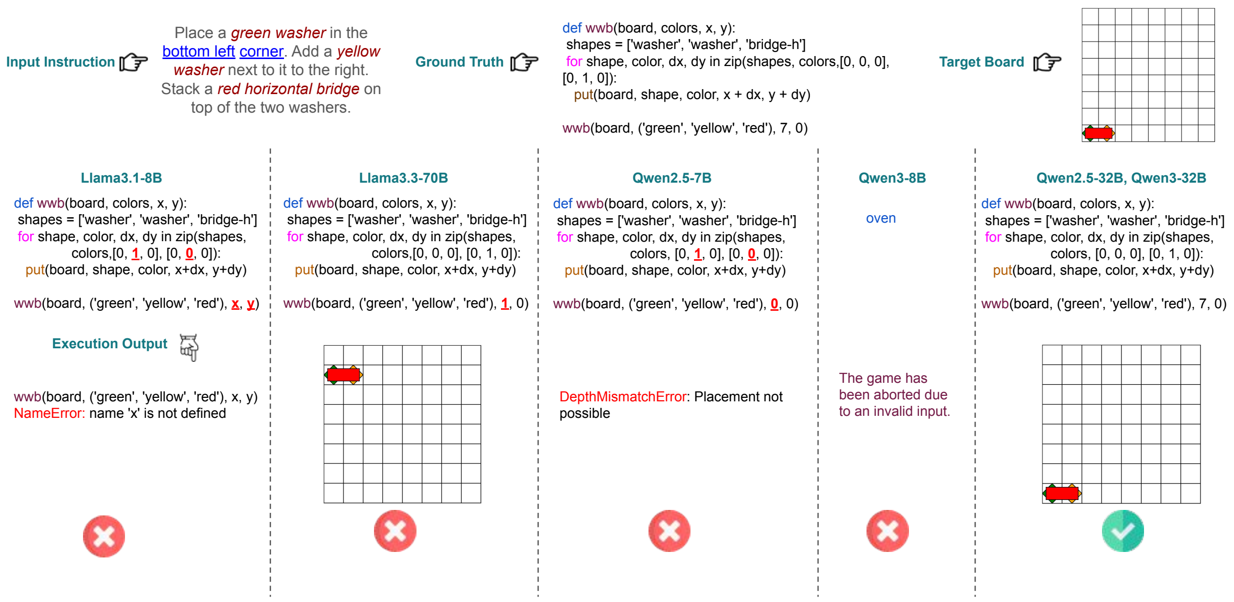
Model	SB		RB	
	ST	HA	ST	HA
Qwen2.5-Coder-7B	0.98	0.40	1.00	0.35
Llama3.1-8B	0.46	0.08	0.75	0.21
Qwen3-8B	0.71	0.09	0.65	0.12
Qwen2.5-Coder-32B	0.98	0.68	1.00	**0.54**
Qwen3-32B	0.99	0.53	1.00	0.38
Llama3.3-70B	0.98	**0.68**	1.00	0.33
GPT-4o	0.78	0.72	0.64	0.55
Claude-4	0.95	0.88	0.98	0.60
Human-Baseline	0.98		**0.76**	

Table 4: Performance by instruction type: ST (synthetic) and HA (human-authored) on SB (simple) and RB (regular) boards.

typically consists of nested `for` loops that repeat the same object placement across multiple locations (see Figure 3). In contrast, simple boards often require placing a sequence of different shapes with precise spatial relationships, typically implemented using abstract function definitions. This output structure appears to be harder for these models to generate reliably.

Effect of prompt variation Additional experiments were conducted using different prompt styles for regular boards. Table 2 shows results from two styles: Function Signature (FSG), which includes only the function signature in the prompt, and Function Schematic (FSC), which includes the signature along with a brief schematic description of the function (see Appendix A.2 for more details). These prompt variations do not significantly affect performance on regular boards, but they reduce performance on simple boards. For example, `Qwen2.5-Coder-32B` drops from 0.68 to 0.45 (FSG) and 0.32 (FSC), while `LLaMA3-70B` drops from 0.69 to 0.54 (FSG) and 0.55 (FSC). These results suggest that while FSG and FSC are sufficient to support symbolic reuse (as in regular boards), they are less effective when the model is required to generate symbolic abstractions, as in simple boards.

Few-shot prompting We also experimented with few-shot prompting using the fine-tuned models. Table 7 (in Appendix A.2.1) shows that few-shot prompting improves `Qwen2.5-Coder-32B`'s score on simple boards to 0.82, but causes a slight decrease in RB performance, from 0.54 to 0.51.



Place a *green washer* in the bottom left corner. Add a *yellow washer* next to it to the right. Stack a *red horizontal bridge* on top of the two washers.

```
def wwb(board, colors, x, y):
 shapes = ['washer', 'washer', 'bridge-h']
 for shape, color, dx, dy in zip(shapes, colors,[0, 0, 0],
[0, 1, 0]):
   put(board, shape, color, x + dx, y + dy)

wwb(board, ('green', 'yellow', 'red'), 7, 0)
```

Llama3.1-8B

```
def wwb(board, colors, x, y):
 shapes = ['washer', 'washer', 'bridge-h']
 for shape, color, dx, dy in zip(shapes,
             colors,[0, 1, 0], [0, 0, 0]):
   put(board, shape, color, x+dx, y+dy)

wwb(board, ('green', 'yellow', 'red'), x, y)
```

```
wwb(board, ('green', 'yellow', 'red'), x, y)
NameError: name 'x' is not defined
```

Llama3.3-70B

```
def wwb(board, colors, x, y):
 shapes = ['washer', 'washer', 'bridge-h']
 for shape, color, dx, dy in zip(shapes,
             colors,[0, 0, 0], [0, 1, 0]):
   put(board, shape, color, x+dx, y+dy)

wwb(board, ('green', 'yellow', 'red'), 1, 0)
```

Qwen2.5-7B

```
def wwb(board, colors, x, y):
 shapes = ['washer', 'washer', 'bridge-h']
 for shape, color, dx, dy in zip(shapes,
             colors, [0, 1, 0], [0, 0, 0]):
   put(board, shape, color, x+dx, y+dy)

wwb(board, ('green', 'yellow', 'red'), 0, 0)
```

```
DepthMismatchError: Placement not
possible
```

Qwen3-8B

```
oven
```

```
The game has
been aborted due
to an invalid input.
```

Qwen2.5-32B, Qwen3-32B

```
def wwb(board, colors, x, y):
 shapes = ['washer', 'washer', 'bridge-h']
 for shape, color, dx, dy in zip(shapes,
             colors, [0, 0, 0], [0, 1, 0]):
   put(board, shape, color, x+dx, y+dy)

wwb(board, ('green', 'yellow', 'red'), 7, 0)
```

Figure 4: Execution analysis of different models on an instruction-to-code generation task. Given a natural language instruction (top left), only `Qwen2.5-32B` and `Qwen3-32B` generate correct and executable code that replicates the target board (top right). Other models fail due to issues such as undefined variables, incorrect coordinates, or invalid placements.

Performance based on Instruction Type We compare model performance on synthetic and human-authored instructions to quantify the generalization gap. Table 4 shows that while models achieve high accuracy on synthetic instructions, performance drops significantly on human-authored inputs. This discrepancy highlights the challenge of transferring symbolic reasoning skills to more abstract and natural instruction styles.

The improved regular board performance seen in `Llama3.1-8B`, `Qwen3-8B` may stem from the relative simplicity of generating repetitive structure, particularly when trained on synthetic instructions that explicitly describe iteration patterns.

Overall, fine-tuning on synthetic data improves performance on regular boards across all models. For simple boards, the impact is varied: some models, particularly instruction-tuned or code-oriented ones, benefit substantially, while others show little improvement due to format violations and hallucinations. These findings indicate that even a small amount of synthetic supervision can be beneficial, but the downstream effects depend on model alignment, size, and prompt structure.

5.2 Qualitative Analysis

Figure 4 presents a qualitative comparison of model predictions for a human-authored instruction. The instruction requires *placing a green washer in the bottom left corner of a grid, adding a yellow washer next to it on the right, and stacking a red horizontal bridge on top of the two washers*. The corresponding ground truth code places the shapes starting from (7, 0) using relative spatial offsets.

`Qwen2.5-32B` and `Qwen3-32B` generated code that exactly matched the target representation. These models correctly interpreted the spatial reference: *bottom left* as (7, 0), maintained the ordering of shape placements, and generated a semantically translated abstract function.

`Llama3-70B` generated syntactically well-formed code with appropriate functional abstraction, but misinterpreted the spatial reference, mapping "bottom left" to (1, 0) rather than (7, 0). `Llama3-8B` shows similar behavior of correct abstraction but incorrectly uses symbolic variables (as x, y) for cell positions. `Qwen2.5-7B` generated correct structure but applied incorrect relative positions, leading to an execution-time placement error. `Qwen3-8B` generated an unrelated response, indicating failure in instruction following. This suggests that while some models are able to replicate correctly, others struggle with grounding language in spatial coordinates.

Error Categorization Each model-generated code was executed, and errors were grouped based on either execution failed or succeeded with a semantic mismatch. Responses that failed to execute were classified as board placement errors and for

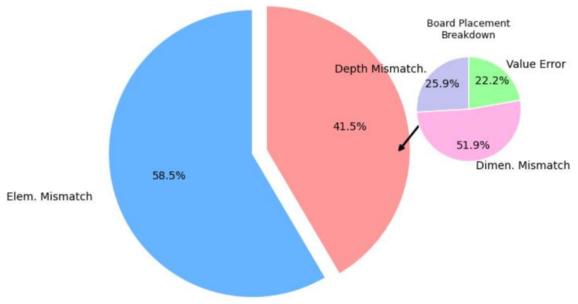


Figure 5: Error distribution on SB (simple) boards for the `Qwen2.5-Coder-32B` model after fine-tuning. The primary pie chart shows the proportion of overall error types, with "Board Placement" accounting for 41.5% and 1 "Element. Mismatch" 58.5%. A secondary pie further decomposes the Board Placement errors into Dimensions mismatch (51.9%), Depth Mismatch (25.9%), and Value Error (22.2%)..

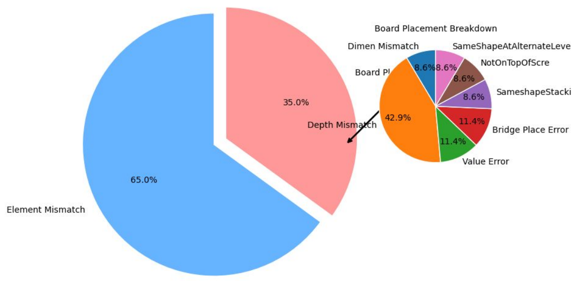


Figure 6: Error distribution on RB (regular) boards for the `Qwen2.5-Coder-32B` model after fine-tuning.

Score Ranges	**BLEU**		**ES**	
	SB	RB	SB	RB
Median Values	0.356	0.024	0.979	0.623
Minimum Values	0.21	0.01	0.92	0.44
Maximum Values	0.66	0.1	0.99	0.82

Table 5: Median and min–max values of instruction similarity (BLEU) and Embedding Similarity (ES) across Simple Boards (SB) and Regular Boards (RB). Lower ES scores on RB highlight increased linguistic variability in human instructions for complex tasks.

successfully executed code, we compare the resulting board with the ground truth and identify mismatches in shape type, color, or placement order. These are classified as element mismatches.

Within board placement errors, we observed different error categories (see Figure 5 and Figure 6). *DepthMismatch* occurs when a bridge is placed without a supporting shape beneath it, typically due to earlier placement errors. *BridgePlacement* errors involve stacking bridges above the allowed two-level height limit. *DimensionMismatch* and *ValueError* arise when object placements exceed board boundaries or use invalid coordinates. Other errors, such as *NameError* and *KeyError*, result from the use of undefined variables or incorrect dictionary access (see Figure 4). Additionally, environment-specific errors such as *NotOnTopOfScrew* and *SameShapeStacking* reflect violations of symbolic constraints defined by the board logic.

Overall, fine-tuning reduced board placement errors across models, with most remaining errors attributable to element mismatches. (more details are in Appendix A.5).

Instruction Similarity Table 1 reports improved model performance on human-authored instructions following fine-tuning, with larger gains observed on simple boards and smaller gains on regular boards. This difference may arise from the phrasing of regular board instructions, which often omit explicit spatial details, making it harder for models to generate the intended code. In contrast, instructions for simple boards are more concrete, even when the corresponding code is complex.

To examine the effect of instruction variation, we compute embedding similarity between synthetic and human-authored instructions for the same board configurations. We use BLEU to measure surface-level overlap and Sentence Transformers (Reimers and Gurevych, 2019) cosine similarity to capture semantic alignment.

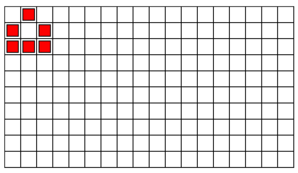
Table 5 presents median and range values across board types. Instructions for simple boards show higher embedding similarity between synthetic and human-authored variants and are associated with larger execution accuracy gains. For regular boards, the similarity is lower and performance drops. To further analyze this effect, we examine how similarity varies with the number of shapes in the object. As shown in Table 9 (in Appendix), instructions referring to objects with fewer shapes tend to have greater execution success. These results suggest that lower semantic and lexical alignment between synthetic and human-authored instructions, particularly in structurally repetitive but abstract cases, limits the generalization ability of models fine-tuned only on synthetic data.

5.3 Cross-Domain Generalization

To assess whether models generalize beyond the vocabulary and structure observed during training, we evaluate models fine-tuned on synthetic data on other datasets.

Transfer to Hexagons The HEXAGONS dataset (Lachmy et al., 2022) consists of human-authored instructions for coloring specific cells on a

Instruction:
Make a red flower, by coloring in red all tiles adjacent to the 2nd tile from the top in the 2nd column from the left



Target Code:
```
for row in range(3):
    for col in range(3):
        if (row == 0 and col == 2) or (row == 1 and col != 1) or
                (row == 2):
            paint(row, col, 'red')
```

Figure 7: Example from the drawing hexagons task with a human-authored instruction and its corresponding target code.

hexagonal grid, based on natural language descriptions. These instructions are similar to repetitive patterns, resembling the regular board structures in our setup. The coloring operation, denoted as `paint(color, row, column)`, is analogous to our `put(shape, color, row, column)` function. The dataset uses a 10×18 hexagonal grid. For evaluation, we use the test split[3], which includes 62 drawing procedures. Table 6 reports model performance on this task before and after fine-tuning. The fine-tuned models show no improvement compared to pretrained models. This limited transferability may be attributed to the domain shift between the training data and the HEXAGONS instructions, which differ in both linguistic style and task structure. Additionally, the larger grid size (10×18) significantly increases the complexity of spatial reasoning and the difficulty of learning accurate coordinate mappings.

Transfer to TidyBot The TIDYBOT dataset (Wu et al., 2023) contains free-form human instructions for real-world object arrangement. The core operation, expressed as `pick_and_place(item, newposition)`, is similar to shape placement in our setting through `put(shape, color, row, column)`. Results are presented in Table 6. Models such as `Qwen2.5-32B` and `LLaMA3-70B` maintain improved task success, demonstrating generalization to other domain instructions and object categories. This suggests that fine-tuning on structured synthetic spatial tasks supports transfer to some domains. In contrast, models such as `Llama3.1-8B`, `Qwen3-8B` show significant performance degradation, indicating limited generalization and potential overfitting to the training domain.

[3] `https://github.com/OnlpLab/Hexagons/blob/main/data/test.jsonl`

# Summary: Put clothes in the laundry basket and toys in the storage box.

Target Code:
```
objects = ["socks", "toy car", "shirt", "Lego brick"]
receptacles = ["laundry basket", "storage box"]
pick_and_place("socks", "laundry basket")
pick_and_place("toy car", "storage box")
pick_and_place("shirt", "laundry basket")
pick_and_place("Lego brick", "storage box")
```

Figure 8: Example task from Tidybot with human-authored instruction and corresponding target code.

Model	HEXAGONS		TIDYBOT	
	Before	After	Before	After
`Qwen2.5-Coder-7B`	0.00	0.03	0.00	0.00
`Llama3.1-8B`	0.00	0.00	0.00	0.039
`Qwen3-8B`	0.00	0.00	0.00	0.00
`Qwen2.5-Coder-32B`	**0.10**	**0.11**	**0.82**	**0.90**
`Qwen3-32B`	0.00	0.03	0.00	0.00
`Llama3.3-70B`	0.06	**0.10**	0.00	0.06
`Human-Baseline`	0.76		0.95	

Table 6: Model performance on HEXAGONS and TIDYBOT tasks before and after fine-tuning.

6 Conclusion

This paper investigates when and how synthetic instruction data can support generalization to human-authored instructions in grounded spatial reasoning tasks. Our findings show that synthetic-only fine-tuning can enable generalization when the arrangement does not involve object repetitions. Through semantic similarity analysis and error categorization, we identified referential ambiguity as a bottleneck: performance declines when instructions contain linguistic variations due to object repetitions. We also observe that regular boards execution success rates see no improvement even with prompt-style variations and few-shot prompting. This suggests that LLMs trained on direct, explicit instruction sets are insufficiently equipped to handle the nuances of compositional abstraction inherent in human-authored instructions. These instructions consists implicit repetition, or structural symmetry posing challenges for models that lack inductive bias toward programmatic generalization. Future work should focus on developing synthetic datasets that mimic these linguistic variations.

Limitations

Our study focuses on generalization from synthetic to human instructions in spatial reasoning tasks, but several limitations remain. First, the training data is entirely synthetic and rule-based; while it enables controlled supervision, it lacks the linguistic diversity, ambiguity, and noise characteristic of real-world language. Second, the target code representation is highly task-specific, designed for grid-based pick-and-place operations, which may limit transferability to other spatial reasoning domains or instruction-following tasks with different semantics. Third, although our evaluation includes human-authored instructions, they are limited to single-turn settings and do not capture the challenges of multi-turn or collaborative spatial interactions.

Acknowledgments

The work reported here has been funded by the Bundesministerium für Bildung und Forschung (BMBF, German Federal Ministry of Research), project "COCOBOTS" (01IS21102A) and Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) grant 423217434 (“RECOLAGE”). We thank the anonymous reviewers for their helpful feedback.

References

Yonatan Bisk, Ari Holtzman, Jesse Thomason, Jacob Andreas, Yoshua Bengio, Joyce Chai, Mirella Lapata, Angeliki Lazaridou, Jonathan May, Aleksandr Nisnevich, Nicolas Pinto, and Joseph P. Turian. 2020. Experience grounds language. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 8718–8735. Association for Computational Linguistics.

Yonatan Bisk, Deniz Yuret, and Daniel Marcu. 2016. 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.

Michael Brenner. 2007. Situation-aware interpretation, planning and execution of user commands by autonomous robots. In *IEEE RO-MAN 2007, 16th IEEE International Symposium on Robot & Human Interactive Communication, August 26-29, 2007, Jeju Island, South Korea, Proceedings*, pages 540–545. IEEE.

Kranti Chalamalasetti, Jana Götze, Sherzod Hakimov, Brielen Madureira, Philipp Sadler, and David Schlangen. 2023. clembench: Using game play to evaluate chat-optimized language models as conversational agents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 11174–11219, Singapore. Association for Computational Linguistics.

Akshay Chaturvedi, Kate Thompson, and Nicholas Asher. 2024. Nebula: A discourse aware Minecraft builder. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 6431–6443, Miami, Florida, USA. Association for Computational Linguistics.

Bao-Anh Dang-Vu, Oliver Porges, and Máximo A. Roa. 2015. Interpreting manipulation actions: From language to execution. In *Robot 2015: Second Iberian Robotics Conference - Advances in Robotics, Lisbon, Portugal, 19-21 November 2015, Volume 1*, volume 417 of *Advances in Intelligent Systems and Computing*, pages 175–187. Springer.

Yan Ding, Xiaohan Zhang, Chris Paxton, and Shiqi Zhang. 2023. Task and motion planning with large language models for object rearrangement. In *IROS*, pages 2086–2092.

Andrew Goldberg, Kavish Kondap, Tianshuang Qiu, Zehan Ma, Letian Fu, Justin Kerr, Huang Huang, Kaiyuan Chen, Kuan Fang, and Ken Goldberg. 2024. Blox-net: Generative design-for-robot-assembly using VLM supervision, physics simulation, and a robot with reset. *CoRR*, abs/2409.17126.

Anders Green, Kerstin Severinson Eklundh, Britta Wrede, and Shuyin Li. 2006. Integrating miscommunication analysis in natural language interface design for a service robot. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2006, October 9-15, 2006, Beijing, China*, pages 4678–4683. IEEE.

Jun Hatori, Yuta Kikuchi, Sosuke Kobayashi, Kuniyuki Takahashi, Yuta Tsuboi, Yuya Unno, Wilson Ko, and Jethro Tan. 2018. Interactively picking real-world objects with unconstrained spoken language instructions. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 3774–3781. IEEE.

Petra Hendriks, Charlotte Koster, and John CJ Hoeks. 2014. Referential choice across the lifespan: Why children and elderly adults produce ambiguous pronouns. *Language, cognition and neuroscience*, 29(4):391–407.

Zichao Hu, Junyi Jessy Li, Arjun Guha, and Joydeep Biswas. 2024. Robo-instruct: Simulator-augmented instruction alignment for finetuning code llms. *arXiv preprint arXiv:2405.20179.*

Chenguang Huang, Oier Mees, Andy Zeng, and Wolfram Burgard. 2023a. Visual language maps for robot navigation. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*, pages 10608–10615. IEEE.

Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. 2023b. Voxposer: Composable 3d value maps for robotic manipulation with language models. In *Conference on Robot Learning, CoRL 2023, 6-9 November 2023, Atlanta, GA, USA*, volume 229 of *Proceedings of Machine Learning Research*, pages 540–562. PMLR.

Helge Hüttenrauch, Kerstin Severinson Eklundh, Anders Green, and Elin Anna Topp. 2006. Investigating spatial relationships in human-robot interaction. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2006, October 9-15, 2006, Beijing, China*, pages 5052–5059. IEEE.

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, and 26 others. 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.

Chenxi Jiang, Chuhao Zhou, and Jianfei Yang. 2025. Rei-bench: Can embodied agents understand vague human instructions in task planning? *CoRR*, abs/2505.10872.

Omkar Joglekar, Shir Kozlovsky, Tal Lancewicki, Vladimir Tchuiev, Zohar Feldman, and Dotan Di Castro. 2024. Towards natural language-driven industrial assembly using foundation models. In *ICLR 2024 Workshop on Large Language Model (LLM) Agents*.

Chalamalasetti Kranti, Sherzod Hakimov, and David Schlangen. 2024a. Retrieval-augmented code generation for situated action generation: A case study on Minecraft. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 11159–11170, Miami, Florida, USA. Association for Computational Linguistics.

Chalamalasetti Kranti, Sherzod Hakimov, and David Schlangen. 2024b. Towards no-code programming of cobots: Experiments with code synthesis by large code models for conversational programming. *CoRR*, abs/2409.11041.

Royi Lachmy, Valentina Pyatkin, Avshalom Manevich, and Reut Tsarfaty. 2022. Draw me a flower: Processing and grounding abstraction in natural language. *Transactions of the Association for Computational Linguistics*, 10:1341–1356.

Zhuoyan Li, Hangxiao Zhu, Zhuoran Lu, and Ming Yin. 2023. Synthetic data generation with large language models for text classification: Potential and limitations. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 10443–10461. Association for Computational Linguistics.

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.

Jonghan Lim, Sujani Patel, Alex Evans, John Pimley, Yifei Li, and Ilya Kovalenko. 2024. Enhancing human-robot collaborative assembly in manufacturing systems using large language models. In *20th IEEE International Conference on Automation Science and Engineering, CASE 2024, Bari, Italy, August 28 - Sept. 1, 2024*, pages 2581–2587. IEEE.

Fangyu Liu, Guy Emerson, and Nigel Collier. 2023. Visual spatial reasoning. *Trans. Assoc. Comput. Linguistics*, 11:635–651.

Annabella Macaluso, Nicholas Cote, and Sachin Chitta. 2024. Toward automated programming for robotic assembly using chatgpt. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 17687–17693. IEEE.

Marvin Minsky. 1980. K-lines: A theory of memory. *Cognitive science*, 4(2):117–133.

Mihai Nadas, Laura Diosan, and Andreea Tomescu. 2025. Synthetic data generation using large language models: Advances in text and code. *CoRR*, abs/2503.14023.

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.

Linus Nwankwo, Bjoern Ellensohn, Ozan Özdenizci, and Elmar Rueckert. 2025. Reli: A language-agnostic approach to human-robot interaction. *arXiv preprint arXiv:2505.01862*.

Rohan Paul, Jacob Arkin, Derya Aksaray, Nicholas Roy, and Thomas M. Howard. 2018. Efficient grounding of abstract spatial concepts for natural language interaction with robot platforms. *Int. J. Robotics Res.*, 37(10).

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.

Lucy Xiaoyang Shi, Brian Ichter, Michael Equi, Liyiming Ke, Karl Pertsch, Quan Vuong, James Tanner, Anna Walling, Haohuan Wang, Niccolo Fusai, Adrian Li-Bell, Danny Driess, Lachy Groom, Sergey Levine, and Chelsea Finn. 2025. Hi robot: Open-ended instruction following with hierarchical vision-language-action models. *CoRR*, abs/2502.19417.

Mohit Shridhar and David Hsu. 2018. Interactive visual grounding of referring expressions for human-robot interaction. In *Robotics: Science and Systems XIV, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, June 26-30, 2018*.

Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. 2023. Progprompt: Generating situated robot task plans using large language models. In *IEEE International Conference on Robotics and Automation, ICRA 2023, London, UK, May 29 - June 2, 2023*, pages 11523–11530. 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.

Karthik Valmeekam, Sarath Sreedharan, Matthew Marquez, Alberto Olmo Hernandez, and Subbarao Kambhampati. 2023. On the planning abilities of large language models (A critical investigation with a proposed benchmark). *CoRR*, abs/2302.06706.

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

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2024a. Voyager: An open-ended embodied agent with large language models. *Trans. Mach. Learn. Res.*, 2024.

Lirui Wang, Yiyang Ling, Zhecheng Yuan, Mohit Shridhar, Chen Bao, Yuzhe Qin, Bailin Wang, Huazhe Xu, and Xiaolong Wang. 2024b. Gensim: Generating robotic simulation tasks via large language models. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

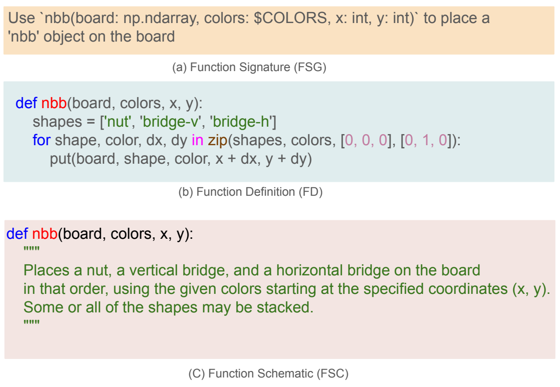
Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg, Szymon Rusinkiewicz, and Thomas A. Funkhouser. 2023. Tidybot: Personalized robot assistance with large language models. In *IROS*, pages 3546–3553.

Mengdi Xu, Peide Huang, Wenhao Yu, Shiqi Liu, Xilun Zhang, Yaru Niu, Tingnan Zhang, Fei Xia, Jie Tan, and Ding Zhao. 2023. Creative robot tool use with large language models. *arXiv preprint arXiv:2310.13065*.

A Appendix

A.1 Prompt Templates

In our task setup, LLMs are prompted to translate natural language instructions into structured code representations. We adopt a chat-style format to align with the pretraining of instruction-tuned models and design a multi-part prompt (see Figure 11



```
Use `nbb(board: np.ndarray, colors: $COLORS, x: int, y: int)` to place a
'nbb' object on the board
```

(a) Function Signature (FSG)

```
def nbb(board, colors, x, y):
    shapes = ['nut', 'bridge-v', 'bridge-h']
    for shape, color, dx, dy in zip(shapes, colors, [0, 0, 0], [0, 1, 0]):
        put(board, shape, color, x + dx, y + dy)
```

(b) Function Definition (FD)

```
def nbb(board, colors, x, y):
    """
    Places a nut, a vertical bridge, and a horizontal bridge on the board
    in that order, using the given colors starting at the specified coordinates (x, y).
    Some or all of the shapes may be stacked.
    """
```

(C) Function Schematic (FSC)

Figure 9: Different styles of prompt variations used for regular board instructions during training and inference.

and Figure 12) that includes details of the 2.5D grid environment along with available API functions. The prompt structure differs slightly between *Simple* and *Regular* boards—particularly in terms of response constraints and the nature of in-context examples provided. This same prompt format is used both during inference and fine-tuning.

A.2 Prompt Variations

To explore how the structure of model prompts affects performance, we introduce three prompt variations for regular board instructions. These variations differ in how the abstract function used in the instruction (e.g., *nbb*) is presented in the prompt context. All three settings include the instruction and the expected output. The difference lies in the prompt information provided about the nbb function. These prompt settings were used consistently across both training and evaluation for each variation. Their impact on model performance is discussed in Section 5.1, and associated results are reported in Table 2.

Function Signature (FSG): The prompt includes only the function signature. This setting simulates the case where the model is expected to use a known abstract function without being shown its implementation. The intent is to assess whether the model can learn to infer the correct usage pattern based solely on its name.

Function Definition (FD): The full definition of the function is included in the prompt. This setting provides the most information about what the function does. It is intended to reduce ambiguity during training and inference and help models correctly ground object placement logic based on a concrete implementation.

Model	ST		HA	
	SB	RB	SB	RB
`Qwen2.5-Coder-7B`	0.95	0.73	0.45	0.20
`Llama3.1-8B`	0.00	0.00	0.00	0.02
`Qwen3-8B`	0.03	0.29	0.01	0.10
`Qwen2.5-Coder-32B`	0.97	0.98	**0.82**	**0.51**
`Qwen3-32B`	0.89	0.52	0.49	0.27
`Llama3.3-70B`	0.73	0.00	0.30	0.07

Table 7: Task performance (accuracy) on simple boards (SB) and regular boards (RB) for synthetic (ST) instructions and human-authored (HA) instructions using few-shot prompting with the Fine-tuned models. Except for `Qwen2.5-Coder-32B` all the models performance was detoriated.

Function Schematic (FSC): The prompt includes the function signature and a schematic docstring describing the function's purpose, but not the actual implementation. This setup serves as an intermediate between FSG and FSD. The schematic is intended to provide semantic guidance without dictating specific implementation. It tests whether models benefit from lightweight documentation, which mirrors how humans often interact with partially known APIs.

A.2.1 Few-shot Prompting

To evaluate whether performance can be further improved with in-context examples, we experimented with few-shot prompting at inference time using the fine-tuned models. For each evaluation instance, we added five synthetic instruction–output pairs of the same board type (simple or regular) to the prompt, followed by the test instruction. The examples were randomly sampled from the training data but excluded any exact match with the test input.

Table 7 reports the results. We observed that few-shot prompting improved performance only for `Qwen2.5-Coder-32B` model and its accuracy is improved to 0.82 (compared to 0.68 in zero-shot). However, the same setting led to a slight performance drop on regular boards, possibly due to increased sensitivity to syntactic variation in human-authored instructions.

A.3 Fine-Tuning

Our dataset consists of spatial instruction-target code pairs over two environment types: *Simple* Boards (SB) and *Regular* Boards (RB). Table 8 summarizes the distribution across training, validation, and test splits for each board type. These splits form the basis for all fine-tuning and evaluation setups described below. All models are fine-tuned using the Unsloth [4] framework with a 4-bit quantization setup to reduce memory overhead. We apply parameter-efficient fine-tuning using LoRA with the following configuration: rank $r = 16$, $\alpha = 16$, dropout=0.0, batch size = 8, learning rate = $1\text{e}{-4}$, and the `adam-8bit` optimizer. This configuration was selected based on preliminary ablation experiments that balanced good trade-off between performance and training efficiency. Fine-tuning is performed for three epochs, and *four* gradient accumulation steps, across all experiments. We use the same prompt template format described in Appendix A.1, ensuring consistency between training and evaluation. All experiments were conducted on a single NVIDIA A100 80GB GPU.

Dataset	**SB**	**RB**	**TB**
`Training Set`	1072	1168	96
`Validation Set`	130	130	-
`Test Set`	130	130	96

Table 8: Dataset statistics showing the number of boards in Simple Boards (SB), Regular Boards (RB), and number of scenarios in TidyBot (TB) across training, validation, and test splits.

A.3.1 Ablation Study

We conducted multiple ablation studies focused on finding the suitable hyper parameter configuration. We begin with identifying the impact of learning rate and batch size, keeping the number of training epochs fixed at 3. We compared learning rates of $1\text{e}{-4}$ and $2\text{e}{-4}$, across batch sizes of 2, 4 and 8. We observed that higher learning rates ($2\text{e}{-4}$) led to faster convergence but sometimes caused overfitting in larger models, particularly when paired with smaller batch sizes. Validation loss curves remained stable for smaller models, but the downstream accuracy did not consistently improve.

To further examine the effect of training duration, we varied the number of epochs while using early stopping (patience 2). We compared runs with 1 epoch, 2 epochs, and 3 epochs (with and without early stopping). Reducing to 1 epoch with a higher learning rate ($2\text{e}{-4}$) led to unstable gains and greater variance across models. Overall, 3 epochs with early stopping of 2 provided more robust results across both board types.

Finally, we evaluated how many training examples were needed to achieve meaningful generalization. We fixed the model and training setup, and

[4]`https://unsloth.ai/`

varied the number of fine-tuning samples (e.g., 100, 300, 500, and 1000). We observed that most models began to generalize reliably to human-authored instructions only when trained on at least 1000 examples (balanced across simple and regular boards). With fewer than 500 examples, performance degraded sharply, especially for regular boards.

A.4 Instruction Similarity vs. Execution Success

We compute similarity between synthetic and human instructions associated with a given board configurations using two measures: BLEU (surface-level overlap) and Embedding cosine similarity using Sentence-BERT (semantic-level alignment).

Table 9 shows how the semantic and lexical similarity between synthetic and human-authored instructions varies with the number of shapes involved in the described object. We observe a clear trend: as the number of shapes per object increases, both lexical and semantic similarity between synthetic and human-written instructions decreases. This decrease is mirrored by a drop in execution success rate (SR).

For simple boards, even though the instructions are often more explicit than those for regular boards, the primary challenge lies in mapping the linguistic description to precise spatial relations and composing an appropriate abstract function. When more shapes are involved, spatial dependencies become more complex, and function construction becomes harder. This leads to a significant decline in execution success, from 1.00 for two-shape objects to 0.29 for five-shape objects.

For regular boards, the instructions often describe repetitive object placement patterns using abstract or minimal phrasing (e.g., "repeat in the next row"). As the number of shapes in the object increases, the underlying pattern logic also becomes more complex, which complicates code generation, particularly when the model must correctly interpret spatial regularities not explicitly stated in the instruction. While the SR remains higher than for simple boards in some cases, the same downward trend is observed.

These findings suggest that both the semantic gap between synthetic and human instructions and the object complexity (in terms of shape count and spatial arrangement) jointly influence model generalization. Models fine-tuned only on synthetic data show limited ability to bridge this gap as object complexity increases.

A.5 Error Categorization

Table 10 presents the percentage of error types observed in model outputs, both before and after fine-tuning. For some models, entries are marked with a dash ('-'), indicating a 100% abort rate and the absence of usable responses for error categorization.

The percentages are computed based on the total number of errors extracted from the model outputs, categorized into Board Placement and Element Mismatch errors.

Before fine-tuning, most models exhibit a higher proportion of Board Placement errors. This is often due to spurious generations violating environmental constraints, such as exceeding spatial boundaries or ignoring depth ordering. After fine-tuning, however, Element Mismatch becomes the dominant error type—suggesting that while structural placement improves, models still struggle to accurately map the intended elements to their correct spatial positions.

The table also helps to qualitatively assess the overall trend in error composition and model behavior across fine-tuning stages.

A.6 Human Evaluation

To establish a human performance baseline, we developed an interactive web interface (Figure 10) that displays the instruction on the left and an editable 8×8 grid on the right. Annotators were tasked with reconstructing the spatial configuration described by the input instruction using the grid. The interface included features such as cell copy-paste to efficiently handle repeated structures and embedded step-by-step guidelines to standardize the reconstruction process.

An annotator was recruited to perform the task. While the annotator had no prior experience in spatial layout tasks, they were familiar with general data labeling workflows. The evaluation involved reconstructing a total of 260 boards (130 *Simple* and 130 *Regular*) and required approximately 30 hours to complete.

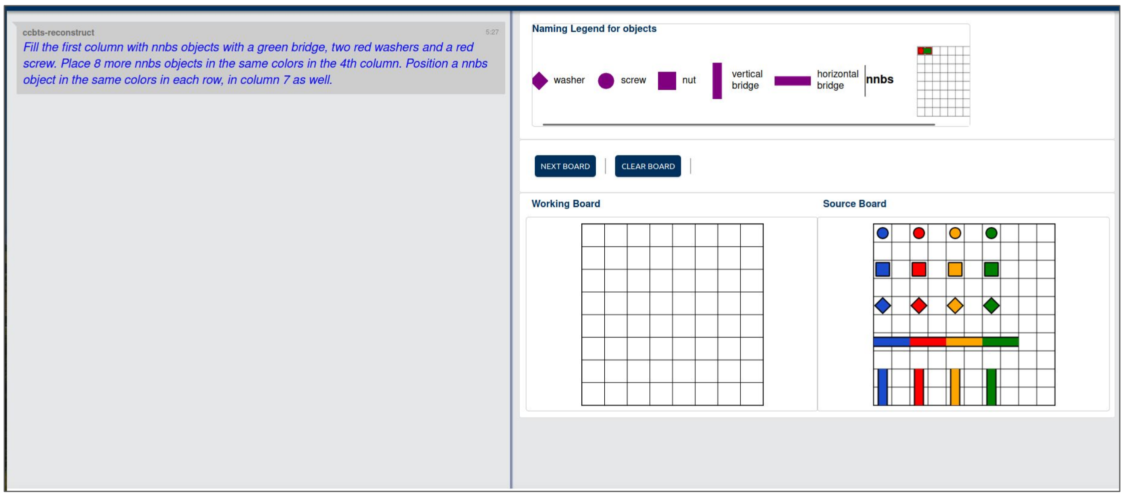
The reconstructed boards were programmatically (by executing the constructed scripts as a result of interaction) compared to the corresponding gold configurations. The resulting match scores are reported as human baseline accuracies and serve as an upper bound reference for model performance.

Shapes Per Object	**SB**			**RB**		
	BLEU	ES	SR	BLEU	ES	SR
2	0.553	0.979	**1.00**	0.033	0.545	**0.80**
3	0.422	0.977	0.90	0.027	0.626	0.43
4	0.327	0.979	0.74	0.024	0.635	0.46
5	0.275	0.979	0.29	0.021	0.671	0.59

Table 9: Instruction similarity and execution success rate (SR) for `Qwen2.5-Coder-32B`, grouped by number of shapes per object. Higher shape counts correlate with reduced lexical and semantic similarity between human and synthetic instructions, and lower execution success.; ES: Embedding Similarity Score;

Model	**Board Placement ↓**				**Element Mismatch ↓**			
	SB		**RB**		**SB**		**RB**	
	Before	After	Before	After	Before	After	Before	After
`Qwen2.5-Coder-7B`	80.17	50.00	–	24.00	19.83	50.00	–	76.47
`Llama3.1-8B`	55.00	61.56	45.00	28.41	45.00	38.46	0.00	71.59
`Qwen3-8B`	–	50.72	100.00	66.67	–	49.28	–	33.33
`Qwen2.5-Coder-32B`	44.64	64.29	33.65	20.00	55.36	35.71	66.35	80.00
`Qwen3-32B`	100.00	50.82	100.00	25.93	–	49.18	–	74.07
`Llama3.3-70B`	78.45	40.00	14.29	16.46	21.56	60.00	85.71	83.54

Table 10: Error Categorization across model responses before and after fine-tuning


ccbts-reconstruct 5:27

Fill the first column with nnbs objects with a green bridge, two red washers and a red screw. Place 8 more nnbs objects in the same colors in the 4th column. Position a nnbs object in the same colors in each row, in column 7 as well.

Figure 10: Web interface used for human reconstruction of spatial configurations. Instructions appear on the left, while the right panel allows annotators to place objects on an 8×8 grid using a visual palette. The interface supports actions such as object selection and cell duplication to aid in building complex or repetitive structures.

TEMPLATE A.6.1

System Info

You are a helpful assistant who is designed to interpret and translate natural language instructions into python executable code snippets.

Environment Info

The environment is an 8x8 grid allowing shape placement and stacking. A shape can be placed in any cell, while stacking involves adding multiple shapes to the same cell, increasing its depth. Shapes typically occupy a single cell, except for the "bridge," which spans two cells and requires two other shapes for stacking. Horizontal bridges span adjacent columns (left and right), and vertical ones span consecutive rows (top and bottom). Stacking is only possible if the shapes have matching depths.

In the grid, columns align with the x-axis and rows with the y-axis. Python indexing is used to identify each cell. The cell in the top-left corner is in the first row and first column, corresponding to x and y values of 0, 0. Similarly, the top-right corner cell is in the first row and eighth column, with x and y values of 0, 7.
- Use the shape name 'bridge-h' if a bridge is placed horizontally
- Use the shape name 'bridge-v' if a bridge is placed vertically

The following functions are already defined; therefore, do not generate additional code for it
- Use ‘put(board: np.ndarray, shape: string, color: string, x: int, y: int) to place a shape on the board

Task Info

For each instruction labeled Instruction: please respond with code under the label Function: followed by a newline and usage for the function under the label Usage: followed by a newline.

Context Info

$INCONTEXT_SAMPLES

Other Info

Do not generate any other text/explanations.

The order of colors, x, y matters, as these are assigned to the shapes in the same sequence. Ensure the response can be executed by Python ‘exec()‘, e.g.: no trailing commas, no periods, etc.
Let's begin

Instruction:
$TEST_INSTRUCTION

Figure 11: Prompt template used for the spatial grounding task for *Simple* boards. 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.6.2

System Info

You are a helpful assistant who is designed to interpret and translate natural language instructions into python executable code snippets.

Environment Info

The environment is an 8x8 grid allowing shape placement and stacking. A shape can be placed in any cell, while stacking involves adding multiple shapes to the same cell, increasing its depth. Shapes typically occupy a single cell, except for the "bridge," which spans two cells and requires two other shapes for stacking. Horizontal bridges span adjacent columns (left and right), and vertical ones span consecutive rows (top and bottom). Stacking is only possible if the shapes have matching depths.

In the grid, columns align with the x-axis and rows with the y-axis. Python indexing is used to identify each cell. The cell in the top-left corner is in the first row and first column, corresponding to x and y values of 0, 0. Similarly, the top-right corner cell is in the first row and eighth column, with x and y values of 0, 7.

- Use the shape name 'bridge-h' if a bridge is placed horizontally - Use the shape name 'bridge-v' if a bridge is placed vertically

The following functions are already defined; therefore, do not generate additional code for it
- Use ‘put(board: np.ndarray, shape: string, color: string, x: int, y: int)‘ to place a shape on the board
- Use ‘$COMBO_NAME(board: np.ndarray, colors: $COLORS, x: int, y: int)‘ to place a '$COMBO_NAME' object on the board

Task Info

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.

The order of colors, x, y matters, as these are assigned to the shapes in the same sequence. Ensure the response can be executed by Python ‘exec()‘, e.g.: no trailing commas, no periods, etc.
Lets begin

Instruction:
$TEST_INSTRUCTION

Figure 12: Prompt template used for the spatial grounding task for *Regular* boards. 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.