

# ROBO-INSTRUCT: SIMULATOR-AUGMENTED INSTRUCTION ALIGNMENT FOR FINETUNING CODE LLMs

Anonymous authors

Paper under double-blind review

## ABSTRACT

Open-weight LLMs are particularly appealing choices to generate training data for fine-tuning Code LLMs on domain-specific service robot applications because they are cost-effective, customizable, and offer better privacy protection. However, unlike proprietary LLMs, open-weight models are more error-prone and often produce programs that violate domain-specific constraints. A promising solution is to incorporate a robot simulator with a well-defined environment to verify program correctness. Yet, these environments require pre-enumeration of relevant entities and their states, which limits the diversity of programs that can be effectively verified. In this work, we introduce ROBO-INSTRUCT that preserves the diversity of programs generated by an LLM while providing the correctness of simulator-based checking. ROBO-INSTRUCT introduces ROBOSIM to *dynamically synthesize consistent simulation environments* for each generated program. Moreover, ROBO-INSTRUCT handles subtler instruction-program inconsistencies that do not result in a constraint violation via INSTALIGN, an LLM-aided instruction-program alignment process. Given domain-specific APIs and a few seed examples, ROBO-INSTRUCT can leverage an 8B Llama3 model to generate a training dataset for fine-tuning a 7B CodeLlama model. Our fine-tuned model achieves a 28.75% improvement in pass@1 over the original base model and a 13.75% improvement compared to its SELF-INSTRUCT-finetuned counterparts, even surpassing the performance of a few proprietary LLMs, such as GPT-3.5-Turbo and Gemini-Pro.

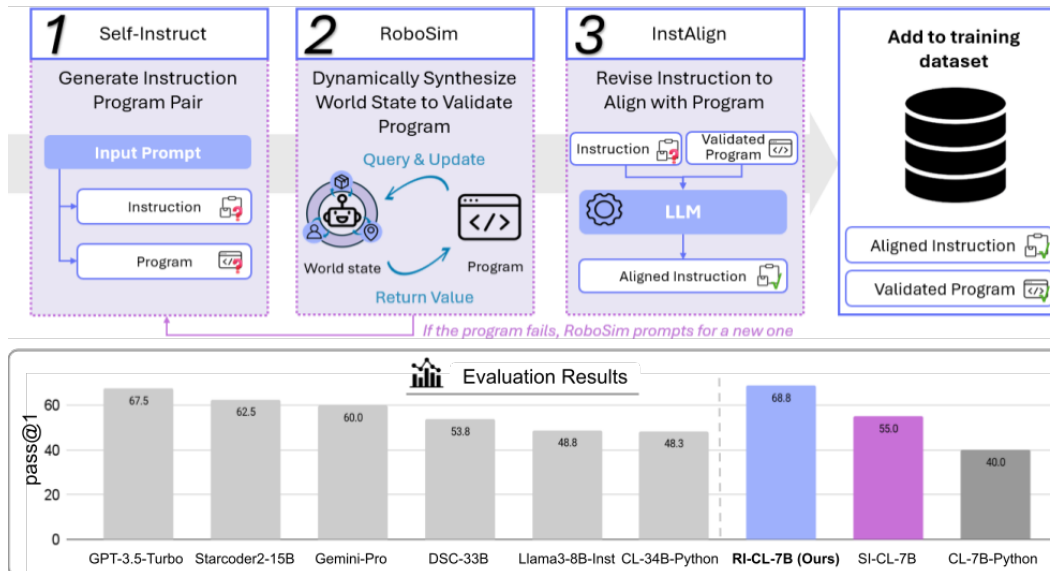


Figure 1: High-level overview of the ROBO-INSTRUCT framework. This figure also shows the pass@1 score performance of ROBO-INSTRUCT fine-tuned LLM compared to other LLMs on ROBOEVAL.

# 1 INTRODUCTION

Large language models (LLMs) have shown great promise at leveraging domain-specific application programming interfaces (APIs) and generating robot programs from natural language instructions (Hu et al., 2024; Huang et al., 2023b; Biggie et al., 2023; Liu et al., 2023a; Wu et al., 2023; Liang et al., 2022; Singh et al., 2023; Huang et al., 2023a). For instance, by formulating a robot’s navigation and perception skills into APIs, such as `go_to(location)` and `is_in_room(object)`, an LLM can generate a program for a service mobile robot to complete the task: “*Determine the number of conference rooms without markers*”. However, despite impressive results, the performance gap remains wide between proprietary and open-weight LLMs in generating robot programs from domain-specific APIs (Hu et al., 2024). To bridge the performance gap, SELF-INSTRUCT (Wang et al., 2022) is a popular method for generating domain-specific data for finetuning LLMs. Given the definition of APIs and a few seed task examples, SELF-INSTRUCT prompts an LLM to generate diverse instruction-program pairs as training data.

This makes SELF-INSTRUCT a seemingly appealing way to prompt open-weight LLMs to generate training data, for the fine-tuning of Code LLMs on domain-specific service robot applications (due to their cost-effectiveness, better privacy protection, and customizability). However, open-weight LLMs are still prone to errors. As a result, using SELF-INSTRUCT naively can produce low-quality data, such as programs that violate domain-specific constraints or instructions that are infeasible for the robot to execute. For instance, as

shown in the figure Fig. 2, the program may attempt to pick up an apple that is not present in the current location (example 1); or the instruction might ask the robot to pick up multiple objects simultaneously, which is physically impossible (example 2). To catch these errors, executing the program in a simulator with well-defined environments may appear promising. However, such environments require pre-enumeration of relevant entities and their states, which can result in rejecting a vast number of programs generated by SELF-INSTRUCT even if they are executable, simply because the pre-defined simulation environment fails to capture certain relevant aspects (e.g., the presence of an “apple”); this undermines the diversity of the data generated.

This work introduces ROBO-INSTRUCT to bring the best of both worlds — it preserves the diversity of programs generated by an LLM while providing the correctness of simulator-based checking. ROBO-INSTRUCT is a new framework based on SELF-INSTRUCT, to address the challenges posed by using an open-weight LLM for generating domain-specific robot programs. As shown in Fig. 1, ROBO-INSTRUCT introduces two novel components: (1) ROBOSIM, a task-agnostic simulator that encodes domain-specific constraints and validates robot programs generated from SELF-INSTRUCT. Critically, ROBOSIM *dynamically synthesizes consistent simulation environments* starting from arbitrary programs. (2) INSTALIGN, an LLM-aided instruction-program alignment procedure that revises the generated instructions to better reflect the intent of the generated programs. ROBO-INSTRUCT also uses a rejection-sampling mechanism to discard invalid programs detected by ROBOSIM and query SELF-INSTRUCT for a new program based on the same instruction.

We use an 8B Llama3 model (AI, 2024) to generate instruction-program training data and fine-tune a 7B CodeLlama model (Rozière et al., 2024) on ROBOEVAL, a domain-specific benchmark for service robot programs. Our ROBO-INSTRUCT-fine-tuned model significantly outperforms the base model by 28.75% in pass@1 scores and achieves a 13.75% improvement over its SELF-INSTRUCT-fine-tuned variant. Moreover, it surpasses several larger code models, including Deepseek-Coder-33B (Guo et al., 2024), Starcoder2-15B (Lozhkov et al., 2024), GPT-3.5-Turbo (OpenAI, 2022), and Gemini-Pro (Team et al., 2024). Finally, in real-world deployment, we demonstrate that ROBO-INSTRUCT achieves significantly lower latency compared with GPT-3.5 and GPT-4.

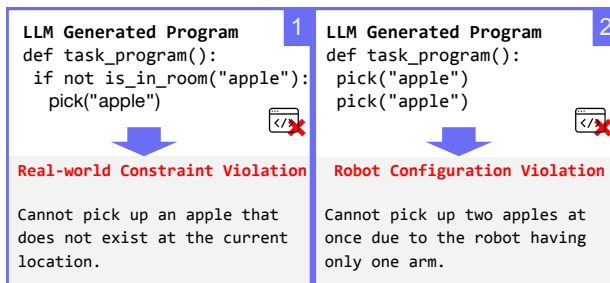


Figure 2: Examples of programs violating domain-specific constraints.

108 **Contributions** Our main contributions are as follows

- 109 1. We introduce ROBO-INSTRUCT, a new framework for improving open-weight LLMs’
- 110 ability to generate training data for finetuning Code LLMs on domain-specific service robot
- 111 applications. This framework introduces two novel components, ROBOSIM and INSTALIGN.
- 112 2. We present ROBOSIM, a method to *synthesize consistent simulation environments dynami-*
- 113 *cally* for verifying the generated programs. This method preserves the diversity of programs
- 114 generated by an LLM while providing correctness with simulator-based checking.
- 115 3. We present INSTALIGN, a simple *instruction-program alignment procedure* that revises
- 116 generated instructions better to reflect the actual results of the generated program.
- 117 4. We show that the ROBO-INSTRUCT finetuned model can significantly outperform the original
- 118 base model and its finetuned variant using SELF-INSTRUCT in generating domain-specific
- 119 service robot programs. It also surpasses several other larger Code LLMs.
- 120

## 121 2 ROBO-INSTRUCT

122 In this section, we describe how ROBO-INSTRUCT generates training datasets for domain-specific

123 service robot programs. Fig. 1 provides a high-level overview of the framework. To add an entry to

124 the training dataset, SELF-INSTRUCT first generates an instruction-program pair based on robot APIs

125 and seed tasks (detailed prompts in Appendix A.4.2). Then, ROBOSIM dynamically synthesizes a

126 consistent simulation environment to validate the program as it executes.

127 If the program is invalid, ROBO-INSTRUCT applies a rejection-sampling method, which generates

128 a new program based on the same instruction and re-evaluates it. This process continues until the

129 program is valid or a predefined maximum resampling limit is reached. If the limit is exceeded, the

130 instruction may be incompatible with the domain-specific APIs or too complex, so the instruction-

131 program pair is discarded.

132 Finally, if the program is valid, INSTALIGN uses an open-weight LLM to refine the instruction to

133 better align with the program’s intent, and the revised instruction-program pair is added to the training

134 dataset. In the following subsections, we delve into the specific design of each component.

### 135 2.1 ROBOSIM: DYNAMIC SYNTHESIS OF SIMULATION ENVIRONMENTS FOR PROGRAM

### 136 VALIDATION

137 We present a principled approach to designing ROBOSIM, a system for dynamically synthesizing

138 consistent simulation environments to validate domain-specific robot programs. For service mobile

139 robots, a simulation environment often relies on three concepts:

- 140 1. A list of *entities* to reason about, e.g., “apple”, “kitchen”
- 141 2. The *type* of the entities, and hence their affordances, e.g., “apple” is an *object*, you can pick
- 142 it up; “kitchen” is a *location*, you can go to it, and it contains objects.
- 143 3. The *state* of the entities in the world, e.g., the “apple” is in the “kitchen”.

144 These concepts are closely related to the domain-specific APIs, where each API invocation during

145 program execution can trigger updates to the simulation environment. To handle the interaction

146 between APIs and simulation environments, we introduce DYNAMICEVAL, an algorithm inspired

147 by Angelic Execution (Broy & Wirsing, 1981), a software engineering technique to infer program

148 properties from incomplete API specifications. DYNAMICEVAL automatically generates a simulation

149 environment for each program and checks its correctness within the inferred environment.

150 As shown in Alg. 1, upon each API invocation, the corresponding inputs and the current simulation

151 environment are passed into DYNAMICEVAL. DYNAMICEVAL first infers relevant entities when

152 they appear in the program being checked (line 5). For instance, if a program includes the statement

153 `pick(“apple”)`, DYNAMICEVAL infers that apple is an entity to consider, even if it is not currently

154 defined in the simulation environment.

155 For each relevant entity, if it has already been initialized (line 7), DYNAMICEVAL infers its type and

156 new state from the API invocation and checks for any inconsistencies with the current simulation

157

158

159

160

161

**Algorithm 1** DYNAMICEVAL( $\text{api\_fn}$ ,  $\text{api\_inputs}$ ,  $\mathcal{W}$ )

---

```

1: Input:  $\text{api\_fn}$  ▷ The API function name
2: Input:  $\text{api\_inputs}$  ▷ The input received by the API invocation
3: Input:  $\mathcal{W}$  ▷ The current simulation environment
4:  $\text{info} \leftarrow \text{EXTRACT\_API\_INVOCATION\_INFO}(\text{api\_fn}, \text{api\_inputs})$ 
5:  $\text{entities} \leftarrow \text{INFER\_RELEVANT\_ENTITIES}(\text{info})$ 
6: for  $\text{entity} \in \text{entities}$  do
7:   if  $\text{IS\_ENTITY\_INITIALIZED}(\text{entity}, \mathcal{W})$  then
8:      $\text{entity\_type} \leftarrow \text{DEDUCE\_TYPE}(\text{info})$ 
9:      $\text{entity\_new\_state} \leftarrow \text{DEDUCE\_STATE}(\text{info})$ 
10:    if  $\text{CHECK\_TYPE\_CONSISTENCY}(\text{entity}, \text{entity\_type}, \text{info}, \mathcal{W})$  and \
11:       $\text{CHECK\_STATE\_CONSISTENCY}(\text{entity}, \text{entity\_new\_state}, \text{info}, \mathcal{W})$  then
12:         $\text{UPDATE\_ENTITY\_STATE}(\text{entity}, \text{info}, \mathcal{W})$ 
13:      else
14:        raise "Error: state inconsistent or type mismatch"
15:      end if
16:    else
17:       $\text{INITIALIZE\_ENTITY\_WITH\_RANDOM\_STATE}(\text{entity}, \text{info}, \mathcal{W})$ 
18:    end if
19:  end for
20:  $\text{retval} \leftarrow \text{GET\_RETURN\_VALUE}(\text{info}, \mathcal{W})$ 
21: return  $\text{retval}$ 

```

---

environment (lines 8-10). For example, `pick` requires an `object` type, while `go_to` requires a `location` type<sup>1</sup>. If a program contains:

---

```

187 1 def task_program():
188 2   pick("apple")
189 3   go_to("apple")

```

---

By invoking API calls sequentially, DYNAMICEVAL first infers that “apple” is an `object` and then raises an error when `go_to(“apple”)` is called. If no inconsistency is detected, the simulation environment is updated accordingly based on the API definition (line 11).

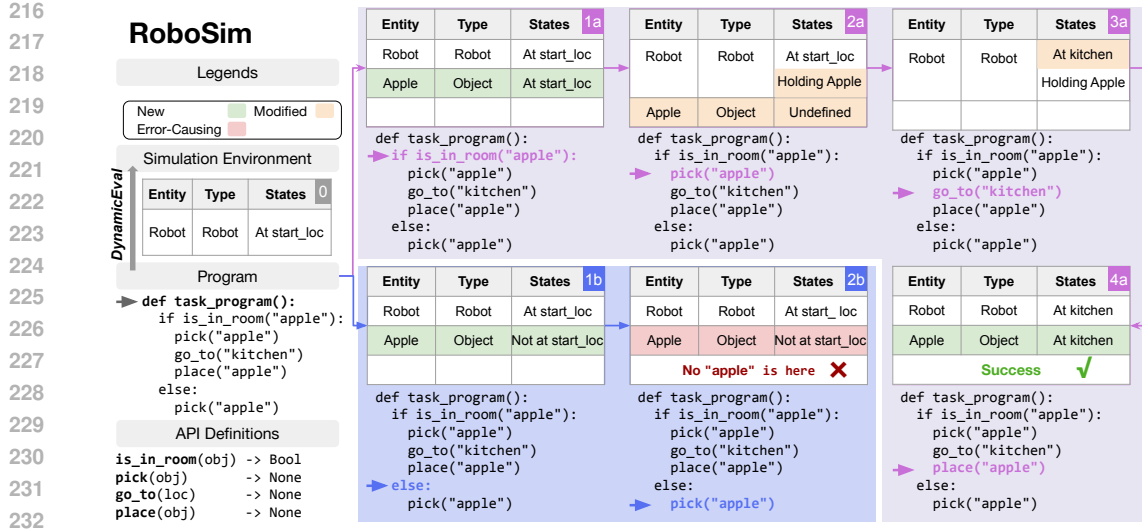
On the other hand, if the entity has not been initialized, it will be assigned a random plausible state (line 16). For example, the API `is_in_room(object)` checks if an object is in the same location as the robot and expects a boolean return. In this case, DYNAMICEVAL assigns a 50% probability for the apple to be in the robot’s current location — determining the state of the apple as either present or absent in the robot’s location.

Finally, DYNAMICEVAL computes and returns the value based on the API specification and the updated simulation environment. Fig. 3 illustrates how ROBOSIM leverages DYNAMICEVAL to dynamically synthesize the simulation environment and validate the program. In ROBOSIM, the simulation starts with only the robot at its initial position, and entities are added as the program runs. At execution timestep 2a, after the robot picks up an apple, it becomes unclear whether another apple remains at the location. Therefore, DYNAMICEVAL sets the apple’s state to “Undefined” and removes it in subsequent executions. This method is also related to STRIPS planning, as we demonstrate the connection in Appendix A.2.

In the example program shown in Fig. 3, it’s clear to humans that the program’s logic is flawed because it attempts to pick up the apple, which is not present in the room. **But how would the simulator identify this as a failing state?** ROBOSIM solves this issue by simulating all possible states of the discovered entities and verifying that none lead to erroneous program execution. In this case, the “apple” can either be present in the room or not. If the apple is not present, executing `pick(“apple”)` will result in an error. Checking all possible states requires exploring an exponentially growing number of combinations based on the entities discovered. To manage this, ROBOSIM

---

<sup>1</sup>In this example, type compatibility check is strict (i.e., “apple” is only an object and no further inference is made about its location). Nevertheless, the algorithm is also capable of handling more advanced scenarios.



233  
234  
235  
236  
237  
238  
239

Figure 3: Illustration of ROBOSIM executing a task program and dynamically updating the simulation environment. Initially, the environment only contains the robot’s starting location. As the program runs, two possible execution paths emerge (purple and blue). The environment is updated by DYNAMICIEVAL at each step, reflecting the robot’s actions, object states, and location changes along both paths.

240  
241  
242

employs a bounded compute budget to randomly sample from this exponential space, as detailed in the overall ROBOSIM algorithm in Alg. 2.

### 243 Algorithm 2 ROBOSIM( $\mathcal{P}$ )

244 **Require:** Program  $\mathcal{P}$  ▷ Generated program  
 245 1: **Initialize:** Set  $\mathcal{A}$  ▷ A set of domain-specific robot APIs  
 246 2: **Initialize:**  $k$  ▷ Number of evaluation iterations  
 247 3: **Initialize:**  $\mathcal{W}_{\text{init}}$  ▷ An initial simulation environment with or without predefined information  
 248 4:  $\mathcal{P}_{\text{trans}} \leftarrow \text{TRANSLATE}(\mathcal{P}, \mathcal{A}, \text{DYNAMICIEVAL})$  ▷ Replace each API call with DYNAMICIEVAL  
 249 5: **for**  $i = 1$  **to**  $k$  **do** ▷ Then, evaluate  $\mathcal{P}$   $k$  times to catch program errors  
 250 6:   **try:**  
 251 7:      $\mathcal{W} \leftarrow \mathcal{W}_{\text{init}}$  ▷ Initialize a new simulation environment  
 252 8:      $\text{exec}(\mathcal{P}_{\text{trans}}, \mathcal{W})$   
 253 9:   **catch:**  
 254 10:    **return** False  
 255 11: **end for**  
 256 12: **return** True ▷ Return True if all program executions are successful

## 257 2.2 INSTALIGN: LLM-AIDED INSTRUCTION-PROGRAM ALIGNMENT PROCEDURE

258  
259  
260  
261  
262

Even after ROBOSIM verifies that a program doesn’t violate domain-specific constraints, it may still have subtle inconsistencies with the instructions — such as omitting a step implied by the instruction. The proposed rejection-sampling strategy alone cannot resolve this issue, and correcting verified programs to fully reflect the instructions while ensuring they remain valid is a challenging task.

263  
264  
265  
266

We note that LLMs nowadays have demonstrated impressive code *understanding* capabilities (Rozière et al., 2024; Nam et al., 2024; Leinonen et al., 2023; Li et al., 2023; Lekshmi-Narayanan et al., 2024). Instead of correcting the program, it may be more effective for the LLM to revise its generated instructions to better align with the program’s intent.

267  
268  
269

We propose INSTALIGN, a procedure that prompts an LLM to revise generated instructions to better match the intent of the program, as shown in Fig. 4. INSTALIGN follows two steps: first, it uses Chain-of-Thought reasoning (CoT) (Wei et al., 2022) to generate a revised instruction based on the instruction-program pair; then, it prompts the LLM to compare the original and revised instructions,

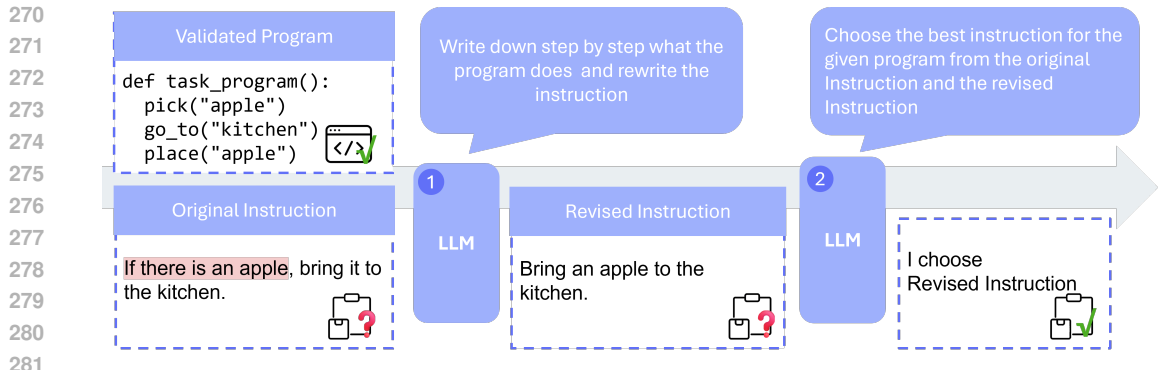


Figure 4: Overview of INSTALIGN.

selecting the one that best aligns with the program’s intent. Detailed prompts are provided in Appendix A.4.3.

### 3 ANALYSIS AND EXPERIMENTS

In this section, we investigate the following two research questions:

1. Is ROBO-INSTRUCT effective at generating training data to fine-tune a small language model to generate domain-specific programs for robots?
2. How do ROBOSIM and InstAlign impact the effectiveness of ROBO-INSTRUCT?

We conduct our investigation by fine-tuning the CodeLlama-Python-7B model (Rozière et al., 2024) on the synthetic dataset generated by ROBO-INSTRUCT and evaluate the fine-tuned model using ROBOEVAL (Hu et al., 2024), a domain-specific code generation benchmark for service mobile robots. In the following subsections, we first provide a brief description of ROBOEVAL. Then we present our experimental results addressing the two main research questions. Finally, we offer more analysis of ROBOSIM, INSTALIGN, and the synthetic dataset.

#### 3.1 ROBOEVAL: A DOMAIN-SPECIFIC ROBOT CODE GENERATION BENCHMARK

**RoboEval Domain-Specific API Definitions**

```

# Get the current location of the robot.
def get_current_location() -> str
# Get a list of all rooms.
def get_all_rooms() -> list[str]
# Check if an object is in the current room.
def is_in_room(object : str) -> bool
# Go to a specific named location.
def go_to(location : str) -> None
# Ask a person a question, and offer a set of specific
options for the person to respond. Returns the response
selected by the person.
def ask(person : str, question : str,
options: list[str]) -> str
# Say the message out loud.
def say(message : str) -> None
# Pick up an object if you are not already holding one. You
can only hold one object at a time.
def pick(obj: str) -> None
# Place an object down if you are holding one.
def place(obj: str) -> None
        
```

**16 RoboEval Benchmark Tasks**

ElevatorTour	FindBackpack	GetDrink	...	WeatherPoll
--------------	--------------	----------	-----	-------------

**Task Instruction**

Go to the elevator. Wait until someone shows up and ask them if they are here for the tour. If yes, welcome them to the university, tell them to follow you, and take them to the main conference room. If not, wait for the next person. When you get to the conference room, say you have arrived at the conference room and also say enjoy your visit here!

**Canonical Solution**

```

1 def task_program():
2   go_to("elevator")
3   while True:
4     if is_in_room("person"):
5       response = ask("", "Are you here for the conference?", ["Yes", "No"])
6       if response == "Yes":
7         say("Welcome to the university. Please follow me.")
8         break
9       time.sleep(1)
10  go_to("conference room")
11  say("We have arrived. Enjoy your time here")
        
```

Figure 5: ROBOEVAL APIs and benchmark task example.

ROBOEVAL is a domain-specific code generation benchmark, featuring a suite of 16 tasks designed to evaluate the ability of LLMs to understand custom APIs and generate programs for service robots. In this domain, a service robot can perceive objects, navigate to various locations, manipulate items, and communicate with humans. Furthermore, the robot should be capable of basic commonsense reasoning and executing complex tasks that involve conditional and repetitive actions. To facilitate these capabilities, ROBOEVAL defines a set of 8 API functions in Python as skill primitives. Fig. 5

illustrates these function signatures and definitions, alongside an example task instruction and its canonical solution from the benchmark. In addition, unlike other popular code generation benchmark tasks (Chen et al., 2021; Austin et al., 2021; Li et al., 2022; Liu et al., 2023b; Lai et al., 2022; Hendrycks et al., 2021), *the order of the robot’s actions is crucial for successfully completing the specified tasks*. For instance, in the task “bring me a marker from the classroom that does not have a whiteboard,” the robot must check each classroom until it finds one without a whiteboard, whereas simply bringing back a marker is insufficient. Hence, ROBOEVAL evaluates the generated program by executing it in a simulator to capture the action traces, which are subsequently validated for sequence correctness using temporal logic.

### 3.2 RQ1: IS ROBO-INSTRUCT EFFECTIVE AT GENERATING TRAINING DATA TO FINE-TUNE A SMALL LANGUAGE MODEL FOR GENERATING DOMAIN-SPECIFIC ROBOT PROGRAMS?

Fine-tune	Model	# Param	ROBOEVAL pass@1		Licensing
			$T = 0$	$T = 0.2$	
-	GPT-4	-	83.75%	85.81%	Proprietary
-	GPT-3.5-Turbo	-	67.5%	65.56%	Proprietary
-	Gemini-Pro	-	60.00%	59.88%	Proprietary
-	Codellama-Python	34B	46.25%	48.25%	Open
-	Starcoder2	15B	62.5%	60.94%	Open
-	Deepseek-Coder	33B	53.75%	52.13%	Open
-	Codellama-Python	7B	40.00%	39.31%	Open
Self-Instruct	CodeLlama-Python	7B	55.00%	52.69%	Open
Robo-Instruct (ours)	CodeLlama-Python	7B	<b>68.75%</b>	<b>66.00%</b>	Open
-	Llama3	8B	42.5%	36.69%	Open
Self-Instruct	Llama-3	8B	55.00%	53.75%	Open
Evol-Instruct	Llama-3	8B	57.5%	54.87%	Open
Robo-Instruct (ours)	Llama-3	8B	<b>66.25%</b>	<b>62.44%</b>	Open
EI + RI (ours)	Llama-3	8B	<b>70.00%</b>	<b>66.38%</b>	Open

Table 1: Pass@1 results of different LLMs on ROBOEVAL computed with greedy decoding  $T = 0$  and nucleus sampling  $T = 0.2$ .

**Experiment Setup.** We use the open-weight LLM, Llama3-8B-Inst, for ROBO-INSTRUCT. To generate a diverse dataset, we employ nucleus sampling for creating instruction-program pairs, setting the temperature  $T = 1$  and top  $p = 0.95$ . The maximum resampling limit is capped at 3 to accommodate instructions that initially produce invalid programs. For the LLM used in INSTALIGN, we empirically adjust the generation temperature to  $T = 0.3$  to optimize performance. Furthermore, we assess the edit similarity between token sequences of each instruction pair in the dataset (Lee et al., 2022), removing duplicates where the similarity score exceeds 0.6. *The same similarity-based approach is used to decontaminate the dataset against the ROBOEVAL benchmark tasks.* We use the same setup to generate data via SELF-INSTRUCT. Instead of discarding invalid programs, SELF-INSTRUCT includes every generated instruction-program pair in the training dataset. Finally, we create two datasets with 5K instruction-program pairs each using SELF-INSTRUCT and ROBO-INSTRUCT respectively. These datasets are then used to fine-tune the CodeLlama-Python-7B model. The learning rate is set to be  $3e-5$  with a warmup ratio of 3% and a constant lr scheduler. We employ the AdamW optimizer (Loshchilov & Hutter, 2019) with an effective batch size of 8, training each model for 5 epochs using a sequence length of 2048 tokens. We train all our models on a single H-100 GPU using unsloth (Unslothai, 2024).

**Baselines.** We divide our baseline models into 2 categories: 1) proprietary LLMs, including GPT4 (OpenAI et al., 2024), GPT3.5-Turbo (OpenAI, 2022), Gemini-Pro (Team et al., 2024), and 2) open-weight LLMs, including Codellama-Python-7B (Rozière et al., 2024), Codellama-Python-34B, Starcoder2-33B (Lozhkov et al., 2024), Deepseek-Coder-33B (Guo et al., 2024), and Llama3-8B-Inst (AI, 2024).

Tab. 1 presents the average pass@1 results for different LLMs on ROBOEVAL, using two settings: greedy decoding at temperature  $T = 0$  and nucleus sampling at temperature  $T = 0.2$ . The results

show that ROBO-INSTRUCT-fine-tuned CodeLlama significantly improves upon the base CodeLlama-Python-7B, and outperforms the SELF-INSTRUCT-fine-tuned variant (Appendix A.3.3 shows that improvements over SELF-INSTRUCT are not the result of distributional biases in the selection process). Notably, ROBO-INSTRUCT surpasses all open-weight models, including larger ones like CodeLlama-Python-34B and Deepseek-Coder-33B. Additionally, although the training dataset was generated using Llama3-8B-Inst, which scores less than 50% pass@1 on ROBOEVAL from Tab. 1, our ROBO-INSTRUCT-fine-tuned model still achieves a significant improvement, scoring 68.75%. Finally, compared to proprietary models, while our ROBO-INSTRUCT-fine-tuned model trails the more powerful GPT-4, it outperforms GPT-3.5-Turbo and Gemini-Pro in generating programs for service mobile robots. This result demonstrates the effectiveness of our approach in generating domain-specific robot program data for fine-tuning a much smaller language model. It suggests that the fine-tuned model could potentially replace some proprietary models, providing a more cost-effective and private option for local deployment.

### 3.3 RQ2: HOW DO ROBOSIM AND INSTALIGN IMPACT THE EFFECTIVENESS OF ROBO-INSTRUCT?

Method	T=0		T=0.2		Invalid Programs
	pass@1	Improv.	pass@1	Improv.	
Codellama-7B-Python	40.00%	+0%	39.31%	+0%	38.31%
SELF-INSTRUCT	55.00%	+15.00%	52.69%	+13.38%	20.94%
+Reject Unsolvable (RU)	60.00%	+20.00%	57.62%	+18.31%	23.38%
+ROBOSIM + RU	63.75%	+23.75%	63.88%	+24.57%	<b>14.13%</b>
+INSTALIGN + RU	58.75%	+18.75%	59.81%	+20.50%	23.44%
<b>+Both (ROBO-INSTRUCT)</b>	<b>68.75%</b>	<b>+28.75%</b>	<b>66.00%</b>	<b>+26.69%</b>	17.07%

Table 2: Pass@1 results of different methods on ROBOEVAL computed with greedy decoding  $T = 0$  and nucleus sampling  $T = 0.2$ . The **Invalid Programs** column indicates the percentage of programs that result in execution errors when tested on ROBOEVAL tasks.

Using the same setup as in the previous section, we investigate the effectiveness of ROBOSIM and INSTALIGN. Since SELF-INSTRUCT may generate instructions for which no corresponding valid program can pass in ROBOSIM, we include Reject Unsolvable (RU) as an additional baseline. SELF-INSTRUCT+RU discards instructions for which no valid programs were found to successfully execute in ROBOSIM, and preserves instructions for which at least one passing program was found. Tab. 2 shows the average pass@1 results from CodeLlama-7B-Python fine-tuned on different datasets generated by each method. First, results from SELF-INSTRUCT + RU indicate that simply discarding invalid instructions improves model performance. Additionally, fine-tuning with a dataset created from SELF-INSTRUCT + ROBOSIM results in the smallest proportion of invalid program errors. Finally, incorporating either ROBOSIM or INSTALIGN individually offers improvements over the baseline SELF-INSTRUCT + RU results, incorporating both in ROBO-INSTRUCT results in the best pass@1 performance. We refer the readers to Appendix A.1 for more results.

### 3.4 QUALITATIVE ANALYSIS OF THE GENERATED PROGRAM ERRORS

We qualitatively analyze invalid programs identified by ROBOSIM, as shown in Fig. 6. The first three examples are easily recognizable to humans as flawed. However, the last example is more complex and involves an error when the robot can navigate to more than two rooms. After the robot places a toy in the living room, DYNAMICIEVAL updates the environment to reflect that a toy is now in the room (line 8). However, when the robot returns to the living room later (line 6), it will not drop the item it’s holding (line 8). As a result, when the robot enters a third room (line 4) and tries to pick up another toy (line 5), an error will occur because the robot is only capable of carrying one item at a time. This example demonstrates that ROBOSIM can detect invalid programs beyond those easily identifiable through human inspection.

<sup>2</sup>Programs have been adapted to succinctly demonstrate the types of errors and fit within the figure.



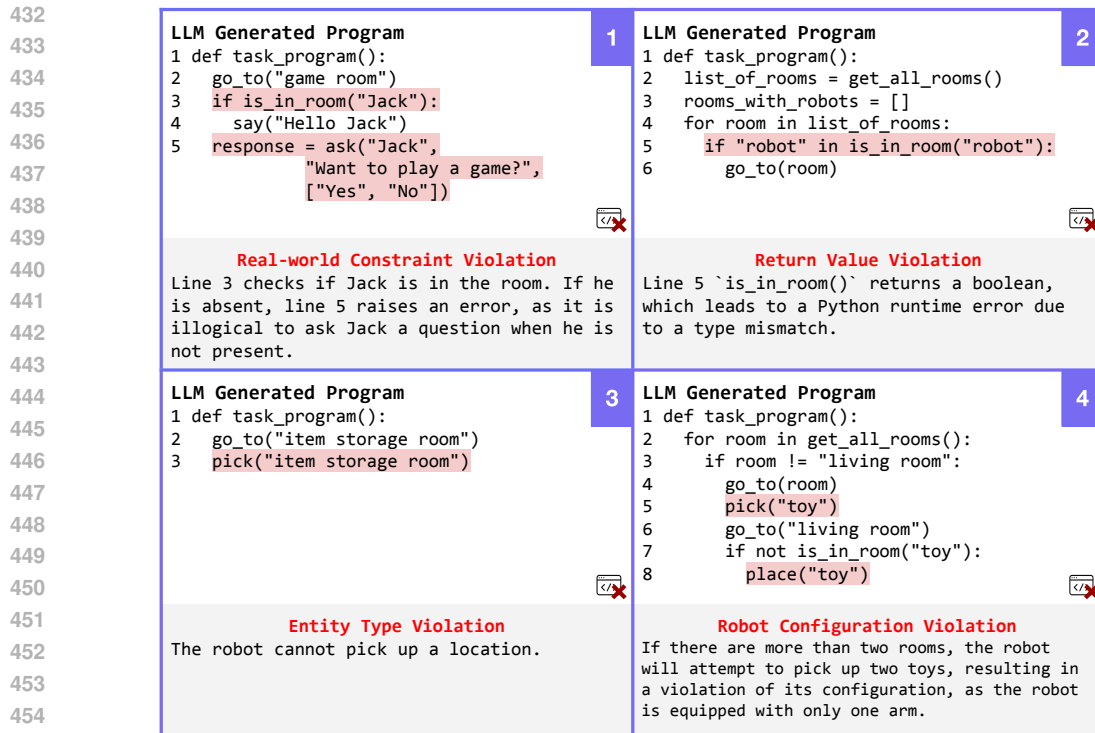


Figure 6: SELF-INSTRUCT-Generated Program Errors. Examples highlight errors that violate domain-specific constraints.<sup>2</sup>

#### 4 REAL-WORLD DEPLOYMENT RESULTS

We deployed the ROBO-INSTRUCT fine-tuned model to generate and execute mobile robot programs in the real world, as shown in Fig. 7. Compared to GPT-4 and GPT-3.5-turbo (Tab. 3), our model generates programs about 6x faster than GPT-4 and 3x faster than GPT-3.5-turbo, with similar output quality. In Appendix A.5 we showcase more results on long-horizon tasks beyond ROBOEVAL.

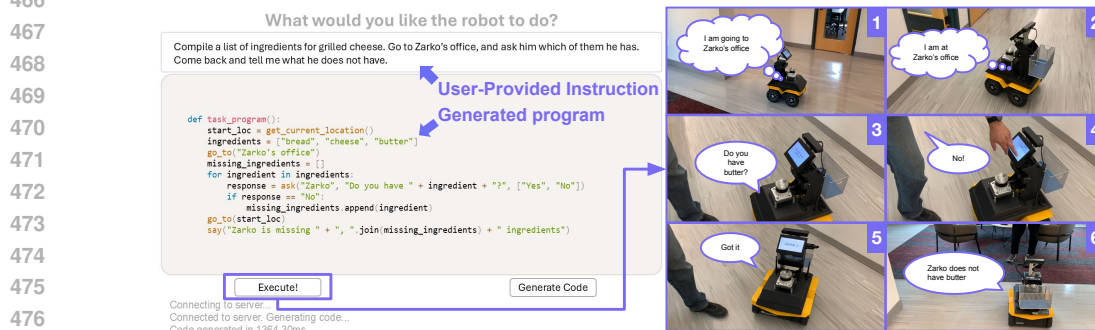


Figure 7: Deployment of the ROBO-INSTRUCT fine-tuned model to generate programs based on user-provided instructions and execute them on the robot.

Models	GPT-4	GPT-3.5-Turbo	Robo-Instruct (Ours)
<b>Inference Speed</b>	19 tokens/s	42 tokens/s	<b>114 tokens/s</b>

Table 3: Inference speed of different models.

## 5 RELATED WORK

### 5.1 LLMs FOR ROBOT CODE GENERATION

LLMs have shown impressive capabilities in generating robot programs from natural language (Liang et al., 2022; Singh et al., 2023; Huang et al., 2023a). One popular approach uses LLMs to generate composable costmaps for robots to plan their motion on. In this approach, Voxposer (Huang et al., 2023b) focuses on the tabletop manipulation setting and NavCon (Biggie et al., 2023) focuses on creating composable maps for navigation. Using LLM to create reward functions is also promising. Eureka (Ma et al., 2023; 2024) and Language to Rewards for Robotic Skill Synthesis (Yu et al., 2023) both show that LLM can generate good reward functions that allows robots to acquire complex skills. Finally, LLM can also be used to generate programs for high-level planning. LLM+p (Liu et al., 2023a) outputs a robot plan in the form of the well-defined planning domain definition language (PDDL). Tidybot (Wu et al., 2023) uses an LLM to generate a rule that captures user preferences from examples and executes a program to sequentially complete the task in order. RoboEval (Hu et al., 2024) focuses on generating domain-specific programs for service mobile robots. It generates a program that allows the service robot to carry out long-horizon tasks and then validates the correctness of the program.

### 5.2 GENERATING DATASETS FOR FINE-TUNING LLMs

To enhance LLMs’ performance in code generation, numerous studies have explored the creation of specialized datasets (Muennighoff et al., 2024; Köpf et al., 2023; Muennighoff et al., 2022). SELF-INSTRUCT (Wang et al., 2022) is one popular method for generating synthetic datasets using an LLM. Following this methodology, Alpaca (Taori et al., 2023) generates 52K instruction-following demonstrations and subsequently fine-tunes the LLaMA 7B model (Touvron et al., 2023) to create Alpaca 7B, which can behave qualitatively similarly to OpenAI’s text-davinci-003. Code Alpaca (Chaudhary, 2023) extends this approach to generate code instructions using 21 seed tasks, while Gorilla-LM (Patil et al., 2023) adapts the method to focus on ML domain-specific APIs from Huggingface, TensorFlow Hub, and Torch Hub. To create more complex instructions, Evol-Instruct (Xu et al., 2024; Luo et al., 2024) proposes iteratively updating instructions to become more complex through different prompting strategies. In addition to Evol-Instruct, OSS-Instruct (Wei et al., 2023) uses open-source code snippets to generate 75K high-quality instruction data and fine-tunes the CodeLlama-Python-7B model to create Magicoder, which can match the performance of GPT-3.5-Turbo (OpenAI, 2022) on HumanEval (Chen et al., 2021). While these works focus on creating seed instruction sets to generate synthetic data to effectively fine-tune an LLM, our research investigates post-processing methods in addition to SELF-INSTRUCT. Specifically, we concentrate on generating domain-specific programs in robotics (Hu et al., 2024), where we can effectively leverage constraints to filter out erroneous programs.

## 6 CONCLUSION, LIMITATION AND FUTURE WORKS

In this work, we introduce ROBO-INSTRUCT, a novel framework to generate synthetic training data to fine-tune small language models for domain-specific robot programs. ROBO-INSTRUCT comprises two novel components: 1) ROBOSIM, a method to synthesize consistent simulation environments dynamically for verifying the generated programs, and 2) INSTALIGN, an LLM-aided instruction alignment procedure to revise instructions to better align with the generated programs. The experimental results show that the 7B CodeLlama model fine-tuned on the ROBO-INSTRUCT dataset significantly outperforms larger open-weight LLMs and proprietary models like GPT-3.5-Turbo and Gemini-Pro in generating service robot programs. However, a key limitation is that ROBO-INSTRUCT relies on SELF-INSTRUCT to filter invalid programs, which may introduce biases and affect dataset quality. Another limitation is the use of a simple rejection-sampling method to handle invalid programs, which may not fully address the underlying issues. Future work will focus on improving dataset quality by integrating ROBO-INSTRUCT with advanced methods like Evol-Inst and OSS-Inst.

## REFERENCES

Meta AI. Introducing meta llama 3: The most capable openly available llm to date. <https://ai.meta.com/blog/meta-llama-3/>, 2024. Accessed: 2024-05-21.

- 540 Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan,  
541 Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large  
542 language models, 2021.
- 543
- 544 Harel Biggie, Ajay Narasimha Mopidevi, Dusty Woods, and Christoffer Heckman. Tell me where to  
545 go: A composable framework for context-aware embodied robot navigation, 2023.
- 546
- 547 Manfred Broy and Martin Wirsing. On the algebraic specification of nondeterministic programming  
548 languages. In *Proceedings of the 6th Colloquium on Trees in Algebra and Programming, CAAP*  
549 '81, pp. 162–179, Berlin, Heidelberg, 1981. Springer-Verlag. ISBN 3540108289.
- 550 Sahil Chaudhary. Code alpaca: An instruction-following llama model for code generation. <https://github.com/sahil280114/codealpaca>, 2023.
- 551
- 552 Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared  
553 Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri,  
554 Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan,  
555 Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian,  
556 Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios  
557 Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino,  
558 Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders,  
559 Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa,  
560 Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob  
561 McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating  
562 large language models trained on code. 2021.
- 563
- 564 Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen,  
565 Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. Deepseek-coder:  
566 When the large language model meets programming – the rise of code intelligence, 2024. URL  
567 <https://arxiv.org/abs/2401.14196>.
- 568
- 569 Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin  
570 Burns, Samir Puranik, Horace He, Dawn Song, and Jacob Steinhardt. Measuring coding challenge  
571 competence with apps. *NeurIPS*, 2021.
- 572
- 573 Zichao Hu, Francesca Lucchetti, Claire Schlesinger, Yash Saxena, Anders Freeman, Sadanand Modak,  
574 Arjun Guha, and Joydeep Biswas. Deploying and evaluating llms to program service mobile robots.  
575 *IEEE Robotics and Automation Letters*, 9(3):2853–2860, 2024. doi: 10.1109/LRA.2024.3360020.
- 576
- 577 Chenguang Huang, Oier Mees, Andy Zeng, and Wolfram Burgard. Visual language maps for robot  
578 navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*  
579 *(ICRA)*, London, UK, 2023a.
- 580
- 581 Wenlong Huang, Chen Wang, Ruohan Zhang, Yunzhu Li, Jiajun Wu, and Li Fei-Fei. Voxposer:  
582 Composable 3d value maps for robotic manipulation with language models. In *7th Annual*  
583 *Conference on Robot Learning*, 2023b.
- 584
- 585 Andreas Köpf, Yannic Kilcher, Dimitri von Rütte, Sotiris Anagnostidis, Zhi Rui Tam, Keith Stevens,  
586 Abdullah Barhoum, Duc Minh Nguyen, Oliver Stanley, Richárd Nagyfi, Shahul ES, Sameer Suri,  
587 David Alexandrovich Glushkov, Arnav Varma Dantuluri, Andrew Maguire, Christoph Schuhmann,  
588 Huu Nguyen, and Alexander Julian Mattick. Openassistant conversations - democratizing large  
589 language model alignment. In *Thirty-seventh Conference on Neural Information Processing*  
590 *Systems Datasets and Benchmarks Track*, 2023. URL <https://openreview.net/forum?id=VSJotgjbPHF>.
- 591
- 592 Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-Tau  
593 Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data  
594 science code generation. *ArXiv*, abs/2211.11501, 2022.
- 595
- 596 Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-  
597 Burch, and Nicholas Carlini. Deduplicating training data makes language models better. In  
598 Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Proceedings of the 60th Annual*

- 594 *Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 8424–8445,  
595 Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.  
596 acl-long.577. URL <https://aclanthology.org/2022.acl-long.577>.  
597
- 598 Juho Leinonen, Paul Denny, Stephen MacNeil, Sami Sarsa, Seth Bernstein, Joanne Kim, Andrew  
599 Tran, and Arto Hellas. Comparing code explanations created by students and large language  
600 models. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer  
601 Science Education V. 1*, ITiCSE 2023, pp. 124–130, New York, NY, USA, 2023. Association  
602 for Computing Machinery. ISBN 9798400701382. doi: 10.1145/3587102.3588785. URL  
603 <https://doi.org/10.1145/3587102.3588785>.
- 604 Arun-Balajjee Lekshmi-Narayanan, Priti Oli, Jeevan Chapagain, Mohammad Hassany, Rabin Banjade,  
605 Peter Brusilovsky, and Vasile Rus. Explaining code examples in introductory programming courses:  
606 Llm vs humans, 2024. URL <https://arxiv.org/abs/2403.05538>.  
607
- 608 Jierui Li, Szymon Tworkowski, Yingying Wu, and Raymond J. Mooney. Explaining competitive-level  
609 programming solutions using llms. *CoRR*, abs/2307.05337, 2023. doi: 10.48550/ARXIV.2307.  
610 05337. URL <https://doi.org/10.48550/arXiv.2307.05337>.
- 611 Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom  
612 Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, Thomas Hubert, Peter Choy, Cyprien  
613 de Masson d’Autume, Igor Babuschkin, Xinyun Chen, Po-Sen Huang, Johannes Welbl, Sven  
614 Gowal, Alexey Cherepanov, James Molloy, Daniel J. Mankowitz, Esme Sutherland Robson,  
615 Pushmeet Kohli, Nando de Freitas, Koray Kavukcuoglu, and Oriol Vinyals. Competition-level code  
616 generation with alphacode. *Science*, 378(6624):1092–1097, 2022. doi: 10.1126/science.abq1158.  
617 URL <https://www.science.org/doi/abs/10.1126/science.abq1158>.  
618
- 619 Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and  
620 Andy Zeng. Code as policies: Language model programs for embodied control. In *arXiv preprint  
621 arXiv:2209.07753*, 2022.
- 622 Bo Liu, Yuqian Jiang, et al. LLM+P: Empowering Large Language Models with Optimal Planning  
623 Proficiency. *arXiv:2304.11477*, 2023a.  
624
- 625 Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by  
626 chatGPT really correct? rigorous evaluation of large language models for code generation. In  
627 *Thirty-seventh Conference on Neural Information Processing Systems*, 2023b. URL <https://openreview.net/forum?id=lqvx610Cu7>.  
628
- 629 Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Confer-  
630 ence on Learning Representations*, 2019. URL <https://openreview.net/forum?id=Bkg6RiCqY7>.  
631  
632
- 633 Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane  
634 Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov,  
635 Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul,  
636 Zhuang Li, Wen-Ding Li, Megan Risdal, Jia Li, Jian Zhu, Terry Yue Zhuo, Evgenii Zheltonozhskii,  
637 Nii Osaе Osaе Dade, Wenhao Yu, Lucas Krauß, Naman Jain, Yixuan Su, Xuanli He, Manan  
638 Dey, Edoardo Abati, Yekun Chai, Niklas Muennighoff, Xiangru Tang, Muhtasham Oblokulov,  
639 Christopher Akiki, Marc Marone, Chenghao Mou, Mayank Mishra, Alex Gu, Binyuan Hui, Tri  
640 Dao, Armel Zebaze, Olivier Dehaene, Nicolas Patry, Canwen Xu, Julian McAuley, Han Hu, Torsten  
641 Scholak, Sebastien Paquet, Jennifer Robinson, Carolyn Jane Anderson, Nicolas Chapados, Mostofa  
642 Patwary, Nima Tajbakhsh, Yacine Jernite, Carlos Muñoz Ferrandis, Lingming Zhang, Sean Hughes,  
643 Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. Starcoder 2 and the stack v2:  
644 The next generation, 2024.
- 645 Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing  
646 Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with  
647 evol-instruct. In *The Twelfth International Conference on Learning Representations*, 2024. URL  
<https://openreview.net/forum?id=UnUwSIgK5W>.

- 648 Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman,  
649 Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding  
650 large language models. *arXiv preprint arXiv: Arxiv-2310.12931*, 2023.
- 651
- 652 Yecheng Jason Ma, William Liang, Hungju Wang, Sam Wang, Yuke Zhu, Linxi Fan, Osbert Bastani,  
653 and Dinesh Jayaraman. Dreureka: Language model guided sim-to-real transfer. 2024.
- 654 Niklas Muennighoff, Thomas Wang, Lintang Sutawika, Adam Roberts, Stella Biderman, Teven Le  
655 Scao, M Saiful Bari, Sheng Shen, Zheng-Xin Yong, Hailey Schoelkopf, et al. Crosslingual  
656 generalization through multitask finetuning. *arXiv preprint arXiv:2211.01786*, 2022.
- 657
- 658 Niklas Muennighoff, Qian Liu, Armel Randy Zebaze, Qinkai Zheng, Binyuan Hui, Terry Yue Zhuo,  
659 Swayam Singh, Xiangru Tang, Leandro Von Werra, and Shayne Longpre. Octopack: Instruc-  
660 tion tuning code large language models. In *The Twelfth International Conference on Learning  
661 Representations*, 2024. URL <https://openreview.net/forum?id=mw1PWNSWZP>.
- 662 Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. Using  
663 an llm to help with code understanding. In *Proceedings of the IEEE/ACM 46th International  
664 Conference on Software Engineering, ICSE '24*, New York, NY, USA, 2024. Association for  
665 Computing Machinery. ISBN 9798400702174. doi: 10.1145/3597503.3639187. URL <https://doi.org/10.1145/3597503.3639187>.
- 666
- 667 OpenAI. Chatgpt: Optimizing language models for dialogue. [https://openai.com/blog/  
668 chatgpt/](https://openai.com/blog/chatgpt/), 2022.
- 669
- 670 OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni  
671 Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor  
672 Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian,  
673 Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny  
674 Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks,  
675 Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea  
676 Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen,  
677 Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung,  
678 Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch,  
679 Damien Deville, et al. Gpt-4 technical report, 2024.
- 680 Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. Gorilla: Large language model  
681 connected with massive apis. *arXiv preprint arXiv:2305.15334*, 2023.
- 682 Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi  
683 Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov,  
684 Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre  
685 Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas  
686 Scialom, and Gabriel Synnaeve. Code llama: Open foundation models for code, 2024.
- 687
- 688 Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter  
689 Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using  
690 large language models. In *2023 IEEE International Conference on Robotics and Automation  
691 (ICRA)*, pp. 11523–11530, 2023. doi: 10.1109/ICRA48891.2023.10161317.
- 692 Karen Sparck Jones. *A statistical interpretation of term specificity and its application in retrieval*, pp.  
693 132–142. Taylor Graham Publishing, GBR, 1988. ISBN 0947568212.
- 694
- 695 Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy  
696 Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model, 2023.
- 697 Gemini Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut,  
698 Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Melvin Johnson,  
699 Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy  
700 Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul R. Barham, Tom  
701 Hennigan, Benjamin Lee, Fabio Viola, et al. Gemini: A family of highly capable multimodal  
models, 2024.

702 Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée  
703 Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand  
704 Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language  
705 models, 2023.

706  
707 Unslothai. Unsloth: Finetune llama 3, mistral & gemma llms 2-5x faster with 80 URL <https://github.com/unslothai/unsloth>. Accessed: 2024-05-22.  
708

709 Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and  
710 Hannaneh Hajishirzi. Self-instruct: Aligning language model with self generated instructions,  
711 2022.

712 Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed H. Chi,  
713 Quoc V Le, and Denny Zhou. Chain of thought prompting elicits reasoning in large language  
714 models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances*  
715 *in Neural Information Processing Systems*, 2022.

716  
717 Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. Magicoder: Source code is  
718 all you need, 2023.

719 Jimmy Wu, Rika Antonova, Adam Kan, Marion Lepert, Andy Zeng, Shuran Song, Jeannette Bohg,  
720 Szymon Rusinkiewicz, and Thomas Funkhouser. Tidybot: Personalized robot assistance with large  
721 language models. *Autonomous Robots*, 2023.

722  
723 Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei  
724 Lin, and Daxin Jiang. WizardLM: Empowering large pre-trained language models to follow  
725 complex instructions. In *The Twelfth International Conference on Learning Representations*, 2024.  
726 URL <https://openreview.net/forum?id=CfXh93NDgH>.

727  
728 Wenhao Yu, Nimrod Gileadi, Chuyuan Fu, Sean Kirmani, Kuang-Huei Lee, Montse Gonzalez Arenas,  
729 Hao-Tien Lewis Chiang, Tom Erez, Leonard Hasenclever, Jan Humplik, Brian Ichter, Ted Xiao,  
730 Peng Xu, Andy Zeng, Tingnan Zhang, Nicolas Heess, Dorsa Sadigh, Jie Tan, Yuval Tassa, and Fei  
731 Xia. Language to rewards for robotic skill synthesis. *Arxiv preprint arXiv:2306.08647*, 2023.

732  
733  
734  
735  
736  
737  
738  
739  
740  
741  
742  
743  
744  
745  
746  
747  
748  
749  
750  
751  
752  
753  
754  
755

## A APPENDIX

### A.1 OVERVIEW

In this appendix, we first outline the relationship between DYNAMICEVAL and the classic STRIPS planning formulation in subsection A.2, providing a new perspective on the proposed algorithm. In subsection A.3, we present additional ablation experiments to analyze the percentage of invalid programs generated by SELF-INSTRUCT and the effectiveness of the rejection-sampling strategy combined with ROBOSIM. We also explore how the generation temperature in INSTALIGN impacts final performance and compare the dataset diversity produced by ROBO-INSTRUCT and SELF-INSTRUCT. Subsection A.4 lists the seed tasks used in ROBOEVAL and the CoT prompts. In subsection A.5, we report real-world experiments that empirically evaluate the performance of our fine-tuned model on two long-horizon tasks, which differ significantly from those in ROBOEVAL, and assess the model’s latency in generating programs. Although this work focuses on service mobile robots, the proposed framework is adaptable to other domains. In subsection A.6, we offer toy examples showing how DYNAMICEVAL can be extended to verify programs by incorporating domain-specific constraints.

### A.2 RELEVANCE TO STRIPS PLANNING

The proposed DYNAMICEVAL shares significant similarities with the formulation of STRIPS planning. A STRIPS instance is typically represented as a tuple  $\langle I, G, A, P \rangle$ , where  $I$  denotes the initial state of the simulation environment,  $G$  represents the desired goal state that the robot aims to achieve,  $A$  defines the set of actions available to transition between states, and  $P$  is the set of preconditions that must be satisfied before performing actions. Thus, DYNAMICEVAL can be reformulated to align with the STRIPS formulation as shown in Alg. 3. Each API invocation corresponds to an action, and its precondition consists of a set of literals, representing specific combinations of entities, types, and states.

To address this, we extend the classic STRIPS formulation by incorporating dynamically discovered literals. Unlike the conventional STRIPS approach, where each literal is binary—True when defined and False when not—we introduce a third value, "Undefined." This means a literal must be explicitly defined as either True or False; otherwise, it remains in the Undefined state. When an action requires a literal that is undefined, a random value (True or False) is assigned to it, and the literal is added to the state of the simulation environment (line 7). Once the precondition is fully defined, the action is executed, and domain-specific constraints are checked for any violations (line 10). This extension enables DYNAMICEVAL to handle arbitrary programs effectively.

---

#### Algorithm 3 DYNAMICEVAL\_STRIPS(api\_fn, params, $\mathcal{W}$ )

---

```

1: Input: api_fn ▷ The API function name
2: Input: api_inputs ▷ The input received by the API invocation
3: Input:  $\mathcal{W}$  ▷ The current state of the simulation environment
4:  $p \leftarrow \text{GETPRECOND}(\text{api\_fn}, \text{params})$  ▷ Get the parameter-specific precondition for api_fn
5: for  $l \in p$  do ▷ Loop through every literal in the precondition
6:   if CHECKDEFINED( $\mathcal{W}, l$ ) is Undefined then
7:      $W \leftarrow \text{GROWWORLD}(l, \mathcal{W})$  ▷ Randomly instantiate the literal and grow  $\mathcal{W}$  to include it
8:   end if
9: end for
10:  $\text{retval}, \mathcal{W} \leftarrow \text{EXECUPDATE}(\text{api\_fn}, \text{params}, \mathcal{W})$  ▷ Execute api_fn and update  $\mathcal{W}$ 
11: return retval

```

---

### A.3 ABLATION EXPERIMENTS

#### A.3.1 THE EFFECTIVENESS OF THE REJECTION-SAMPLING STRATEGY

We analyze the percentage of instruction-program pairs discarded by ROBOSIM at various maximum resampling limits, as shown in Fig. 8. Initially, with the maximum resampling limit set to 0, disabling the rejection-sampling method, approximately 51% of the programs generated by SELF-INSTRUCT contain errors. As the limit increases, fewer programs are discarded. However, there is a diminishing

810  
811  
812  
813  
814  
815  
816  
817  
818  
819  
820  
821  
822  
823  
824  
825  
826  
827  
828  
829  
830  
831  
832  
833  
834  
835  
836  
837  
838  
839  
840  
841  
842  
843  
844  
845  
846  
847  
848  
849  
850  
851  
852  
853  
854  
855  
856  
857  
858  
859  
860  
861  
862  
863

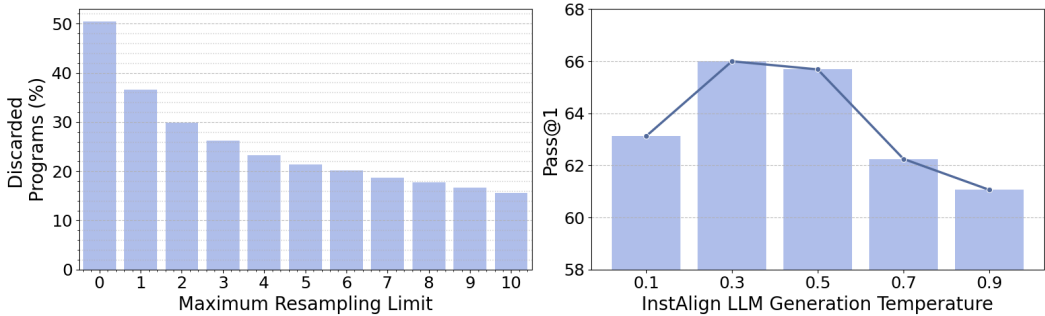


Figure 8: Ablation Experiment Results

return; even with the maximum resampling limit set to 10, about 15% of the instructions still result in invalid programs.

### A.3.2 INSTRUCTION ALIGNMENT MODEL TEMPERATURE

We further investigate how varying LLM temperatures for generating the revised instruction in INSTALIGN impact the performance of the fine-tuned model. Fig. 8 shows the bar chart of the pass@1 score of the models fine-tuned over datasets generated using different LLM temperatures in INSTALIGN. The model performs the best when fine-tuned on the dataset generated using LLM temperature  $T = 0.3$ . As the temperature increases, we observe a decrease in performance.

### A.3.3 ANALYSIS OF GENERATED DATASET

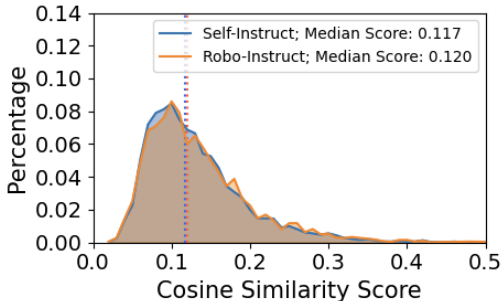


Figure 9: Cosine similarities between ROBOEVAL and generated data.

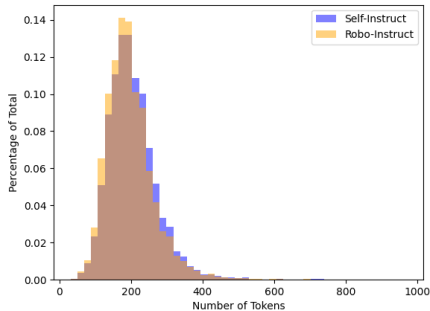


Figure 10: Token length distribution for each instruction-program pair.

Similar to Magicoder (Wei et al., 2023), we show the improvements from ROBO-INSTRUCT are not merely due to selection bias, i.e., including data more aligned with the distribution of ROBOEVAL tasks than SELF-INSTRUCT. We pair each sample from the generated dataset with task instructions and their canonical solutions, then compute cosine similarity using TF-IDF embeddings (Sparck Jones, 1988). Fig. 9 shows comparable token similarities between both methods. Additionally, Fig. 10 presents the token length distribution, which also appears similar for both.

Method	Size	Ngram=4 Score	# Synth. Loc.	# Synth. Obj.
SELF-INSTRUCT	5K	0.581	956	1060
ROBO-INSTRUCT	5K	0.587	1025	928

Table 4: Dataset Statistics

Since ROBOSIM does not rely on pre-defined simulation environments, we aim to assess the diversity of programs generated by SELF-INSTRUCT and whether ROBO-INSTRUCT can maintain this diversity.



To do so, we measure the number of distinct entities, such as synthetic locations and objects. As shown in Tab. 4, with a dataset of only 5,000 samples, approximately 1,000 unique objects and locations are generated, highlighting that conventional robot simulations with pre-defined environments are insufficient. Additionally, Tab. 4 presents the n-gram diversity scores for each dataset, indicating that both distributions and dataset statistics are highly similar. This suggests that ROBO-INSTRUCT not only preserves but enhances the quality of generated data compared to SELF-INSTRUCT, rather than simply aligning the dataset with benchmark tasks.

## A.4 PROMPTS

### A.4.1 ROBOEVAL SEED TASK EXAMPLE

Seed Task Example 1:

---

```

878 1 # Instruction: Go to Arjun's office, ask him if he is ready to head out,
879 2 # and come back and tell me what he said
880 3 def task_program():
881 4     start_loc = get_current_location()
882 5     go_to("Arjun's office")
883 6     response = ask("Arjun",
884 7                 "Are you ready to go?",
885 8                 ["Yes", "No"])
886 9     go_to(start_loc)
887 10    say("Arjun said: " + response)

```

---

Seed Task Example 2:

---

```

890 1 # Instruction: Ask Alice if she needs 1, 2, or 3 boxes.
891 2 # Go to the storage room and ask if they have that many boxes.
892 3 # If so, go place the boxes in Alice's office.
893 4 # Otherwise, tell Alice you could not get the boxes.
894 5 def task_program():
895 6     go_to("Alice's office")
896 7     num_boxes = ask("Alice",
897 8                 "How many boxes do you need?",
898 9                 ["1", "2", "3"])
899 10    go_to("storage room")
900 11    response = ask("",
901 12                "Do you have" + num_boxes + " boxes?",
902 13                ["Yes", "No"])
903 14    if response == "Yes":
904 15        for _ in range(int(num_boxes)):
905 16            pick("box")
906 17            go_to("Alice's office")
907 18            place("box")
908 19            go_to("storage room")
909 20    else:
910 21        go_to("Alice's office")
911 22        say("I could not get the boxes")

```

---

Seed Task Example 3:

---

```

913 1 # Instruction: Check if there is a red marker in the main
914 2 # office, and if so, tell Eve that there is a marker there.
915 3 # If not, go to the supply room and
916 4 # bring a red marker to the main office.
917 5 def task_program():
918 6     go_to("main office")
919 7     red_marker_found = is_in_room("red marker")

```

```

918 8     if red_marker_found:
919 9         go_to("Eve's office")
920 10        say("There is a red marker in the main office")
921 11     else:
922 12        go_to("supply room")
923 13        pick("red marker")
924 14        go_to("main office")
925 15        place("red marker")

```

---

## Seed Task Example 4:

```

926
927
928
929 1 # Instruction: Check every classroom if there is a whiteboard.
930 2 # Go to Aiden's office to tell him which room does not
931 3 # have a whiteboard. Come back and tell me task is completed.
932 4 def task_program():
933 5     start_loc = get_current_location()
934 6     list_of_rooms = get_all_rooms()
935 7     room_without_whiteboard = []
936 8     for room in list_of_rooms:
937 9         if "classroom" not in room:
938 10            continue
939 11            go_to(room)
940 12            if not is_in_room("whiteboard"):
941 13                room_without_whiteboard.append(room)
942 14            go_to("Aiden's office")
943 15            if len(room_without_whiteboard) > 0:
944 16                message = ""
945 17                for room in room_without_whiteboard:
946 18                    message += room + ", "
947 19                message += "do not have a whiteboard"
948 20            else:
949 21                message = "all classrooms have a whiteboard"
950 22            say(message)
951 23            go_to(start_loc)
952 24            say("task is completed")

```

---

## Seed Task Example 5:

```

953 1 # Instruction: Go to the kitchen and wait for someone
954 2 # to show up. When someone shows up, ask them to open
955 3 # the fridge, then pick up a diet coke.
956 4 # Finally, put the diet coke in the living room.
957 5 def task_program():
958 6     go_to("kitchen")
959 7     while True:
960 8         if is_in_room("person"):
961 9             response = ask("",
962 10                "Please open the fridge",
963 11                ["Yes", "No"])
964 12             if response == "Yes":
965 13                 pick("diet coke")
966 14                 break
967 15             time.sleep(1)
968 16            go_to("living room")
969 17            place("diet coke")

```

---

## Seed Task Example 6:

```

970
971 1 # Instruction: Take a bed sheet from the laundry room
972 2 # and put it in each of the bedrooms.

```

```

972 3 def task_program():
973 4     start_loc = get_current_location()
974 5     list_of_rooms = get_all_rooms()
975 6     for room in list_of_rooms:
976 7         if "bedroom" not in room:
977 8             continue
978 9         go_to("laundry room")
979 10        pick("bed sheet")
980 11        go_to(room)
981 12        place("bed sheet")
982 13    go_to(start_loc)

```

---

#### 983 A.4.2 PROMPTS TO GENERATE SYNTHETIC DATASET USING SELF-INSTRUCT

```

984
985
986 1 You are a helpful assistant. Here is a robot that has the
987 2 following capabilities:
988 3 - def get_current_location() -> str:
989 4 - def get_all_rooms() -> list[str]:
990 5 - def is_in_room(object : str) -> bool:
991 6 - def go_to(location : str) -> None:
992 7 - def ask(person : str, question : str, options: list[str]) -> str:
993 8 - def say(message : str) -> None:
994 9 - def pick(obj: str) -> None:
995 10 - def place(obj: str) -> None:
996 11 Generate an interesting robot task that can be accomplished using the
997 12 above capabilities.
998 13 {SEED EXAMPLE 1}
999 14 ...
1000 15
1001 16
1002 17 Generate an interesting robot task that can be accomplished using the
1003 18 above capabilities.
1004 19 {SEED EXAMPLE 6}
1005 20
1006 21 Generate an interesting robot task that can be accomplished using the
1007 22 above capabilities.

```

---

#### 1008 A.4.3 COT PROMPTS FOR INSTALIGN

```

1009 1 ### Role
1010 2 You are an expert at understanding robot programs.
1011 3 You will be given a task instruction and robot program pair.
1012 4 However, the instruction may not align with the program well.
1013 5 You need to correct the task instruction to match the given robot program.
1014 6
1015 7 ### Context
1016 8 The robot only has access to the following 8 APIs and
1017 9 standard Python functions
1018 10 - def get_current_location() -> str:
1019 11 - def get_all_rooms() -> list[str]:
1020 12 - def is_in_room(object : str) -> bool:
1021 13 - def go_to(location : str) -> None:
1022 14 - ask(person : str, question : str, options: list[str]) -> str:
1023 15 - say(message : str) -> None:
1024 16 - def pick(obj: str) -> None:
1025 17 - def place(obj: str) -> None:
1026 18
1027 19 ### Inputs
1028 20 Original Instruction
1029 21 This is a task instruction that may not align with the robot program
1030 22 Robot Program

```

```

1026     This is a python function starting with `def task_program():`
1027
1028     """ Task
1029     1. Write down all the provided APIs used in the program and
1030     explain the effect of each API in this program
1031     2. Examine these APIs and write down step by step what the program does
1032     3. Combine all the results above and rewrite the instruction
1033     You need to be specific and clear in your final corrected instruction.

```

---

## 1035 A.5 REAL-WORLD EXPERIMENTS

### 1037 A.5.1 PROGRAM GENERATION RESULTS BEYOND ROBOEVAL

1038 We conducted a small qualitative experiment to assess the performance of the base model, SELF-  
 1039 INSTRUCT fine-tuned models, ROBO-INSTRUCT fine-tuned models, and GPT-4 on long-horizon tasks.  
 1040 To showcase performance, we created two new long-horizon tasks that differ significantly from those  
 1041 in the ROBOEVAL benchmark. Using a temperature setting of 0, we generated programs with each  
 1042 model and found that our ROBO-INSTRUCT fine-tuned model successfully produced correct programs,  
 1043 while both the base model and SELF-INSTRUCT fine-tuned model failed. Additionally, GPT-4 made  
 1044 an error on the second instruction. We show the generation results from ROBO-INSTRUCT fine-tuned  
 1045 models and GPT-4 below.

#### 1046 Task instruction 1:

```

1049 1 Let's play a game: Double and give it to the next person. Start with
1050 2 1 dollar. Go to rooms A, B, C, D, E, F, and G. If you see someone, tell
1051 3 them how much money you have. Then ask if they would like to take the
1052 4 money now or double the amount and give it to the next person. If they
1053 5 choose to take it, the game is over, and you should come back to me.
1054 6 Otherwise, double your money and continue. If, in the end, no one takes
1055 7 the money, tell me how much you still have.

```

---

#### 1056 ROBO-INSTRUCT fine-tuned model (Ours) [Success]

```

1059 1 def task_program():
1060 2     start_loc = get_current_location()
1061 3     list_of_rooms = ["A", "B", "C", "D", "E", "F", "G"]
1062 4     money = 1
1063 5     for room in list_of_rooms:
1064 6         go_to(room)
1065 7         if is_in_room("person"):
1066 8             say("I have $" + str(money))
1067 9             response = ask("", ""Would you like to take the money now
1068 10             or double the amount and give it to the next person?""",
1069 11             ["Take now", "Double and give"])
1070 12             if response == "Take now":
1071 13                 go_to(start_loc)
1072 14                 say("The game is over, I gave the money to " + room)
1073 15                 return
1074 16             else:
1075 17                 money *= 2
1076 18             go_to(start_loc)
1077 19             say("No one took the money, I still have $" + str(money))

```

---

#### 1076 GPT-4 [Success]

```

1079 1 def task_program():
1080 2     start_loc = get_current_location()

```

```

1080     rooms = ["A", "B", "C", "D", "E", "F", "G"]
1081     money = 1
1082     for room in rooms:
1083         go_to(room)
1084         if is_in_room("person"):
1085             say("I have " + str(money) + " dollars")
1086             response = ask("", ""Would you like to take the money now
1087             or double the amount and give it to the next person?""",
1088             ["Take it", "Double it"])
1089             if response == "Take it":
1090                 say("Game over. The person in room " + room +
1091                 " took the money.")
1092                 go_to(start_loc)
1093                 return
1094             else:
1095                 money *= 2
1096         go_to(start_loc)
1097         say("No one took the money. I still have " + str(money) + " dollars.")

```

---

### Task instruction 2:

Go to my office and check if I have a table, a chair, and a monitor there. If any of these items are missing, go to Jason's office and see if he is there. If he is, ask him if I can borrow the missing items. If he agrees, pick up each missing item and bring it to my office. If Jason is not in his office or he says no, come back and tell me the reason.

---

### ROBO-INSTRUCT fine-tuned model (Ours) [Success]

```

1109 def task_program():
1110     start_loc = get_current_location()
1111     go_to("my office")
1112     table_found = is_in_room("table")
1113     chair_found = is_in_room("chair")
1114     monitor_found = is_in_room("monitor")
1115     if not table_found:
1116         go_to("Jason's office")
1117         if is_in_room("Jason"):
1118             response = ask("Jason", "Can I borrow the table from you?",
1119             ["Yes", "No"])
1120             if response == "Yes":
1121                 go_to("Jason's office")
1122                 pick("table")
1123                 go_to("my office")
1124                 place("table")
1125             else:
1126                 go_to(start_loc)
1127                 say("Jason said no")
1128         else:
1129             go_to(start_loc)
1130             say("Jason is not in his office")
1131     if not chair_found:
1132         go_to("Jason's office")
1133         if is_in_room("Jason"):
1134             response = ask("Jason", "Can I borrow the chair from you?",
1135             ["Yes", "No"])
1136             if response == "Yes":
1137                 go_to("Jason's office")
1138                 pick("chair")
1139                 go_to("my office")

```

```

1134         place("chair")
1135     else:
1136         go_to(start_loc)
1137         say("Jason said no")
1138     else:
1139         go_to(start_loc)
1140         say("Jason is not in his office")
1141 if not monitor_found:
1142     go_to("Jason's office")
1143     if is_in_room("Jason"):
1144         response = ask("Jason", "Can I borrow the monitor from you?",
1145             ["Yes", "No"])
1146         if response == "Yes":
1147             go_to("Jason's office")
1148             pick("monitor")
1149             go_to("my office")
1150             place("monitor")
1151         else:
1152             go_to(start_loc)
1153             say("Jason said no")
1154     else:
1155         go_to(start_loc)
1156         say("Jason is not in his office")
1157 go_to(start_loc)
1158 say("task is completed")

```

---

#### GPT-4 [Failure]

```

1159 def task_program():
1160     start_loc = get_current_location()
1161     go_to("my office")
1162     missing_items = []
1163     for item in ["table", "chair", "monitor"]:
1164         if not is_in_room(item):
1165             missing_items.append(item)
1166     if len(missing_items) > 0:
1167         go_to("Jason's office")
1168         if is_in_room("Jason"):
1169             for item in missing_items:
1170                 response = ask("Jason", "Can I borrow your " + item +
1171                     "?", ["Yes", "No"])
1172                 if response == "Yes":
1173                     # BUG: did not return to Jason's office after going back to my office
1174                     pick(item)
1175                     go_to("my office")
1176                     place(item)
1177                 else:
1178                     go_to(start_loc)
1179                     say("Jason did not allow me to borrow his " + item)
1180                 return
1181         else:
1182             go_to(start_loc)
1183             say("Jason is not in his office")
1184         return
1185     go_to(start_loc)
1186     say("Task is completed")

```

---

1183  
1184  
1185  
1186  
1187

1188  
1189  
1190  
1191  
1192  
1193  
1194  
1195  
1196  
1197  
1198  
1199  
1200  
1201  
1202  
1203  
1204  
1205  
1206  
1207  
1208  
1209  
1210  
1211  
1212  
1213  
1214  
1215  
1216  
1217  
1218  
1219  
1220  
1221  
1222  
1223  
1224  
1225  
1226  
1227  
1228  
1229  
1230  
1231  
1232  
1233  
1234  
1235  
1236  
1237  
1238  
1239  
1240  
1241

## A.5.2 GENERATION LATENCY

What would you like the robot to do?

Go to the elevator. Wait until someone shows up and ask them if they are here for the tour. If yes, welcome them to the university, tell them to follow you, and take them to the main conference room. If not, wait for the next person. When you get to the conference room, say you have arrived at the conference room and also say enjoy your visit here!

```
def task_program():
    start_loc = get_current_location()
    while True:
        go_to("elevator")
        if is_in_room("person"):
            response = ask("", "Are you here for the tour?", ["Yes", "No"])
            if response == "Yes":
                say("Welcome to the university! Please follow me.")
                go_to("main conference room")
                break
            time.sleep(1)
        say("I have arrived at the conference room. Enjoy your visit here!")
        go_to(start_loc)
```

Execute! Generate Code

Connecting to server...  
Connected to server. Generating code...  
Code generated in 1162.00ms.

Generated by our model

(a) Example Program 1 generated by our fine-tuned model.

What would you like the robot to do?

Go to the elevator. Wait until someone shows up and ask them if they are here for the tour. If yes, welcome them to the university, tell them to follow you, and take them to the main conference room. If not, wait for the next person. When you get to the conference room, say you have arrived at the conference room and also say enjoy your visit here!

```
def task_program():
    go_to("elevator")
    while True:
        if is_in_room("person"):
            response = ask("", "Are you here for the tour?", ["Yes", "No"])
            if response == "Yes":
                say("Welcome to the university! Please follow me to the main conference room.")
                go_to("main conference room")
                say("We have arrived at the main conference room. Enjoy your visit here!")
                break
            time.sleep(1)
```

Execute! Generate Code

Connecting to server...  
Connected to server. Generating code...  
Code generated in 6822.60ms.

Generated by GPT-4

(b) Example Program 1 generated by GPT-4.

Figure 11: Our fine-tuned model is approximately 6x faster in inference speed than GPT-4 (Part 1).

1242  
1243  
1244  
1245  
1246  
1247  
1248  
1249  
1250  
1251  
1252  
1253  
1254  
1255  
1256  
1257

**What would you like the robot to do?**


Go to the elevator. Wait until someone shows up and ask them if they are here for the tour. If yes, welcome them to the university, tell them to follow you, and take them to the main conference room. If not, wait for the next person. When you get to the conference room, say you have arrived at the conference room and also say enjoy your visit here!

```
def task_program():
    go_to("elevator")
    while True:
        if is_in_room("person"):
            response = ask("", "Are you here for the tour?", ["Yes", "No"])
            if response == "Yes":
                say("Welcome to the university! Please follow me to the main conference room.")
                go_to("main conference room")
                say("We have arrived at the main conference room. Enjoy your visit here!")
                break
            time.sleep(1)
```

Execute!
Generate Code

Connecting to server...  
Connected to server. Generating code...  
Code generated in 6822.60ms.

Generated by GPT-4



(c) Another Example Program generated by GPT-4.

1260  
1261  
1262  
1263  
1264  
1265  
1266  
1267  
1268  
1269  
1270  
1271  
1272  
1273  
1274  
1275  
1276

**What would you like the robot to do?**


Go to the elevator. Wait until someone shows up and ask them if they are here for the tour. If yes, welcome them to the university, tell them to follow you, and take them to the main conference room. If not, wait for the next person. When you get to the conference room, say you have arrived at the conference room and also say enjoy your visit here!

```
def task_program():
    go_to("elevator")
    while True:
        if is_in_room("person"):
            response = ask("", "Are you here for the tour?", ["Yes", "No"])
            if response == "Yes":
                say("Welcome to the university! Please follow me to the main conference room.")
                go_to("main conference room")
                say("We have arrived at the main conference room. Enjoy your visit here!")
                break
            time.sleep(1)
```

Execute!
Generate Code

Connecting to server...  
Connected to server. Generating code...  
Code generated in 6822.60ms.

Generated by GPT-4



(d) Another Example Program generated by GPT-4.

Figure 12: Our fine-tuned model is approximately 6x faster in inference speed than GPT-4 (Part 2).

## 1280 A.6 TOY EXAMPLES BEYOND SERVICE MOBILE ROBOTS

### 1281 A.6.1 ROBOT WITH LOW-LEVEL CONTROLS

1284 Consider a tabletop manipulation scenario with a potential API function, `is_rotate(robot_gripper_name, radians)`, where the robot's gripper has a physical constraint, allowing rotation only within the range  $[-\frac{\pi}{6}, \frac{\pi}{6}]$  radians. For the following generated program snippet:

```
1288
1289 1 def task_program():
1290 2     rotate("left hand", math.pi/6)
1291 3     rotate("left hand", math.pi/6)
1292 4     rotate("left hand", math.pi/6)
```

1293  
1294 DYNAMICIEVAL will first infer that "left hand" is an entity of the robot gripper type. Then, regardless  
1295 of the initial configuration of the gripper, DYNAMICIEVAL will throw an error because the program causes the gripper to exceed its allowable range of motion.



## 1296 A.6.2 AI-POWERED PERSONAL DIGITAL ASSISTANT

1297

1298 Consider a broader application than robotics: code generation for an AI-powered personal digital  
1299 assistant. This AI assistant could handle scheduling events using an API function like `schedule_on_calendar(event, start_time, duration)`. Given the instruction: *"My schedule is free tomorrow morning. Please create two 1-hour timeslots for office hours for my robotics and deep learning class."*

1301

1302

1303

1304

1305

1306

1307

1308

1309

1310

1311

1312

1313

1314

1315

1316

1317

1318

1319

1320

1321

1322

1323

1324

1325

1326

1327

1328

1329

1330

1331

1332

1333

1334

1335

1336

1337

1338

1339

1340

1341

1342

1343

1344

1345

1346

1347

1348

1349

The assistant could generate a program to create these timeslots:

---

```
1304 1 def task_program():  
1305 2     schedule_on_calendar("robotics class office hour",  
1306 3                          "9:30 am", "1 hr")  
1307 4     schedule_on_calendar("deep learning class office hour",  
1308 5                          "10:00 am", "1 hr")
```

---

In this example, DYNAMICEVAL needs to reason about the entities “robotics class office hour” and “deep learning class office hour”, which are categorized as event types. The event type indicates that these entities have associated timeslots. The state of these entities is defined by the time they occur: robotics class office hour is set for 9:30-10:30 am, and deep learning class office hour is set for 10:00-11:00 am. During evaluation, DYNAMICEVAL can identify a time conflict between these two office hours and thus determine that the generated program is invalid.