Deploying and Evaluating LLMs to Program Service Mobile Robots

Zichao Hu[1], Francesca Lucchetti[2], Claire Schlesinger[2], Yash Saxena[1], Anders Freeman[3], Sadanand Modak[1], Arjun Guha[2], Joydeep Biswas[1]

[1]Department of Computer Science, University of Texas at Austin
[2]Khoury College of Computer Sciences, Northeastern University
[3]Department of Computer Science, Wellesley College

Introduction

We are interested in deploying service mobile robots to perform arbitrary user tasks from natural language descriptions. Recent advancements in large language models (LLMs) have shown promise in related applications involving visuomotor tasks (Liang et al. 2022; Singh et al. 2023; Huang et al. 2023), planning (Ahn et al. 2022; Huang et al. 2022; Driess et al. 2023; Liu et al. 2023), and in this work, we investigate the use of LLMs to generate programs for *service mobile robots* leveraging mobility, perception, and human interaction skills, where accurate *sequencing and ordering of actions* is crucial for success. We present CODEBOTLER and ROBOEVAL (Hu et al. 2023): CODEBOTLER is an open-source robot-agnostic tool to generate general-purpose service robot programs from natural language, and ROBOEVAL is a benchmark for evaluating LLMs' capabilities of generating programs to complete service robot tasks.

While the capabilities of LLMs at producing robot programs are impressive, they are still susceptible to a variety of failures. To understand the nature of the failures, we need an effective method to evaluate these programs. Existing benchmarks typically rely on either simple input-output unit test functions (Chen et al. 2021; Liang et al. 2022), or they utilize complex high-fidelity 3D simulations (Singh et al. 2023). However, checking for input-output pairs is insufficient when it comes to evaluating service robot programs. Consider the task *"Check how many conference rooms have no markers"*. It is insufficient to just check for the number of conference rooms stated by the LLM-generated programs. Rather, the correctness of the program depends on *the sequence of robot actions taken*. In this example, the robot must visit each conference room and check for markers, before arriving at the final answer. Furthermore, the correct sequence of actions may depend on the specific *world state*. Finally, we observe that there are significant variations in the correctness of generated programs with small variations in the phrasing of the natural language task descriptions (Babe et al. 2023).

We thus introduce the ROBOEVAL benchmark to address these challenges in evaluating LLM-generated programs for service mobile robots. This benchmark integrates three key components: a symbolic simulator, a trace evaluator, and a comprehensive suite of 16 tasks. Fig. 1 shows the system diagram of CODEBOTLER and ROBOEVAL. When a program created by CODEBOTLER is passed into ROBOEVAL, it undergoes a two-step evaluation process. First, the program is executed within the symbolic simulator, which produces multiple program traces corresponding to different initial world states. Next, these traces are evaluated against a set of temporal specifications. These specifications are designed to define and affirm the correct behavior for the given task, tailored to each specific initial world state.

The use of the symbolic simulator and trace evaluator in ROBOEVAL makes the evaluation both efficient and precise. This approach not only reduces the computational load but also accurately captures the necessary sequence of robot actions. This method provides a more realistic and practical assessment of LLM-generated robot programs, particularly in validating their sequence logic and operational viability.

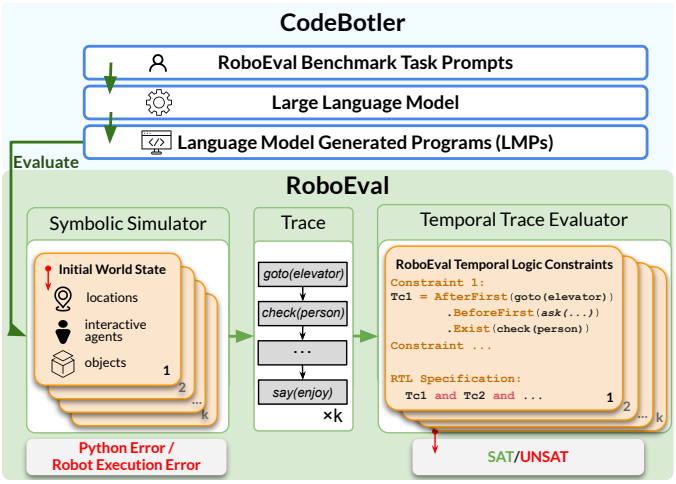

Figure 1: The system diagram of CODEBOTLER and ROBOEVAL. CODEBOTLER receives a task prompt and queries a large language model (LLM) to generate a robot program. Then ROBOEVAL evaluates the generated programs using a symbolic simulator and a temporal trace evaluator to determine whether each program satisfies the task constraints or not.

AAAI 2024 Spring Symposium on User-Aligned Assessment of Adaptive AI Systems. Stanford University, Stanford, CA, USA.

We used the RoboEval benchmark to evaluate and analyze the performance of five state-of-the-art Large Language Models (LLMs). The models tested include: 1. GPT-4 (OpenAI 2023), 2. GPT-3.5 (Brown et al. 2020) (`text-davinci-003`), and 3. PaLM2 (Anil et al. 2023) (`text-bison-001`) as state-of-the-art API-only proprietary models; and 4. CodeLlama (Rozière et al. 2023) (`Python-34b-hf`) and 5. StarCoder (Li et al. 2023) as open-access models. Our analysis categorizes the types of failures exhibited by these different LLMs when benchmarked against RoboEval. Common failure categories identified include Python runtime errors, errors in executing infeasible robot actions, and failures in meeting specific task requirements.

For additional information and detailed insights into our study, we refer the reader to our project website: https://amrl.cs.utexas.edu/codebotler/.

RoboEval Results

To gain insights into the capabilities and limitations of different state-of-the-art LLMs for generating service mobile robot LMPs, we use the RoboEval benchmark to empirically answer the following questions:

1. First, how do different LLMs perform in generating programs for tasks in the RoboEval benchmark?
2. Second, when a generated service robot LMP fails, what are the causes?

Performance Of LLMs On The RoboEval Benchmark

The RoboEval benchmark consists of 16 tasks, each with 5 prompt paraphrases, totaling 80 different prompts. For each prompt, we generate 50 program completions and calculate the pass@1 score (Chen et al. 2021), a common metric for LMP evaluation. This score indicates the probability of an LMP being correct if an LLM generates only one LMP for a given prompt.

We compute the percentage of prompts that have a pass@1 score greater than or equal to a threshold value, which ranges from 1 to 0. We present this information in Fig. 2 as a Cumulative Distribution Function (CDF). Although relaxing the pass@1 score threshold for each LLM increases prompt coverage, there are still certain prompts (ranging from 48.75% for StarCoder to 1.25% for GPT-4) where LLMs consistently fail to generate correct LMPs.

Causes of Failures of LMPs

We evaluate the failure modes of the LMPs and classify these failures into three categories: 1. Python Errors, including syntax, runtime, and timeout errors; 2. Robot Execution Errors, that occurs when a program attempts to execute an infeasible action, such as navigating to a non-existent (hallucinated) location; and 3. Task Completion Errors, where the program runs correctly in the simulator but fails RTL checks for task completion. We use RoboEval's symbolic simulator to detect and classify Python Errors and Robot Execution Errors, and we use RoboEval's evaluator to capture the Task Completion Errors. Fig. 3shows the breakdown of these failure categories for each LLM.

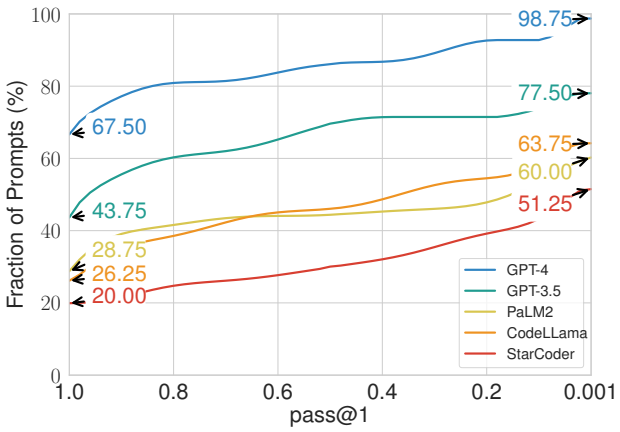

Figure 2: Cumulative Distribution Function (CDF) curves depict the percentage of prompts for which each LLM can generate correct LMPs at various pass@1 score thresholds. A perfect LLM would show a horizontal line at 100%, indicating it can generate correct LMPs for all prompts with a pass@1 score of 1. To maintain visual clarity, we limit the x-axis to 10^{-3} since all CDF plots eventually reach 100%.

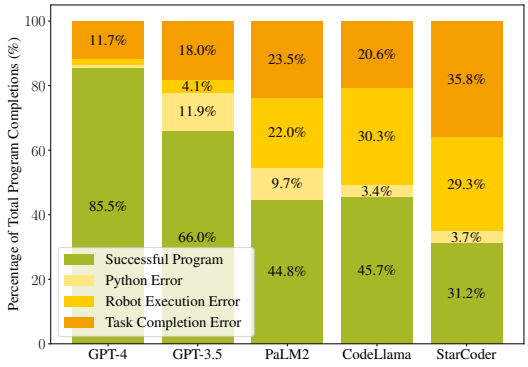

Figure 3: Causes of failures for LMPs on the RoboEval benchmark.

We observe that despite having fewer parameters, the CodeLLMs (CodeLlama and StarCoder) generally make fewer Python errors. This suggests that LLMs trained on a larger proportion of code may be more adept at generating successful completions in the DSL defined in the prompt.

For a detailed analysis of our study, please refer to our paper available on arXiv at https://arxiv.org/pdf/2311.11183.pdf. This paper offers an in-depth look at our methodologies, findings, and their implications in the field of LLM-guided robot program generation.

References

Ahn, M.; Brohan, A.; Brown, N.; et al. 2022. Do As I Can, Not As I Say: Grounding Language in Robotic Affordances. arXiv:2204.01691.

Anil, R.; Dai, A. M.; Firat, O.; et al. 2023. PaLM 2 Technical Report. arXiv:2305.10403.

Babe, H. M.; Nguyen, S.; Zi, Y.; Guha, A.; Feldman, M. Q.; and Anderson, C. J. 2023. StudentEval: A Benchmark of Student-Written Prompts for Large Language Models of Code. *arXiv:2306.04556*.

Brown, T. B.; Mann, B.; Ryder, N.; et al. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165.

Chen, M.; Tworek, J.; Jun, H.; Yuan, Q.; et al. 2021. Evaluating large language models trained on code. *arXiv:2107.03374*.

Driess, D.; Xia, F.; Sajjadi, M. S. M.; et al. 2023. PaLM-E: An Embodied Multimodal Language Model. arXiv:2303.03378.

Hu, Z.; Lucchetti, F.; Schlesinger, C.; Saxena, Y.; Freeman, A.; Modak, S.; Guha, A.; and Biswas, J. 2023. Deploying and Evaluating LLMs to Program Service Mobile Robots. arXiv:2311.11183.

Huang, W.; Abbeel, P.; Pathak, D.; and Mordatch, I. 2022. Language Models as Zero-Shot Planners: Extracting Actionable Knowledge for Embodied Agents. arXiv:2201.07207.

Huang, W.; Wang, C.; Zhang, R.; Li, Y.; Wu, J.; and Fei-Fei, L. 2023. VoxPoser: Composable 3D Value Maps for Robotic Manipulation with Language Models. arXiv:2307.05973.

Li, R.; Allal, L. B.; Zi, Y.; et al. 2023. StarCoder: may the source be with you! arXiv:2305.06161.

Liang, J.; Huang, W.; Xia, F.; Xu, P.; Hausman, K.; Ichter, B.; Florence, P.; and Zeng, A. 2022. Code as Policies: Language Model Programs for Embodied Control. In *arXiv:2209.07753*.

Liu, B.; Jiang, Y.; Zhang, X.; Liu, Q.; Zhang, S.; Biswas, J.; and Stone, P. 2023. LLM+P: Empowering Large Language Models with Optimal Planning Proficiency. arXiv:2304.11477.

OpenAI. 2023. GPT-4 Technical Report. arXiv:2303.08774.

Rozière, B.; Gehring, J.; Gloeckle, F.; et al. 2023. Code Llama: Open Foundation Models for Code. arXiv:2308.12950.

Singh, I.; Blukis, V.; Mousavian, A.; Goyal, A.; Xu, D.; Tremblay, J.; Fox, D.; Thomason, J.; and Garg, A. 2023. ProgPrompt: Generating Situated Robot Task Plans using Large Language Models. In *ICRA 2023*.