Code-as-Control: Just-in-Time Policy Generation and Editing for Robotic Arms via Large Language Models

Anish Kalra

***Abstract*—Robotic manipulation systems traditionally rely on pre-defined skill libraries, hand-authored policies, or large-scale end-to-end learned models trained on extensive datasets. While recent work has shown that large language models (LLMs) can assist with task planning or high-level program composition, most approaches either restrict the model to selecting from existing actions or require significant retraining and data collection. In this work, we explore Code-as-Control, a paradigm in which an off-the-shelf LLM generates executable, low-level robotic control policies in real time from natural language commands, using only a thin, low-context API exposed to the model. We integrate Gemini-2.5 Pro and Gemini-3 Pro with a Sawyer robotic arm in simulation, enabling the model to synthesize Cartesian and joint-space motion policies on demand. We further introduce a natural-language policy editing loop, allowing users to iteratively refine previously generated behaviors without rewriting code manually. To evaluate this capability, we propose the Robotic Gestures Benchmark (RGB), which tests multi-step sequencing, trajectory generation, timing, control structure usage, and edit efficiency. Our results show a substantial performance gap between Gemini-2.5 and Gemini-3, with the latter achieving perfect task success and requiring significantly fewer corrective interactions. These findings suggest that modern LLMs can serve as effective just-in-time policy generators for robotic control, offering a lightweight alternative to both skill-based systems and data-intensive end-to-end models.**

I. Introduction

Programming robotic manipulation behaviors remains a challenging and time-intensive process. Traditional approaches require either extensive hand-engineering of task-specific controllers or the construction of large libraries of reusable skills. While these methods can be reliable, they significantly limit flexibility and impose a high barrier to rapid prototyping. More recently, end-to-end learning approaches have demonstrated impressive generalization capabilities, but at the cost of massive data collection, specialized training infrastructure, and limited interpretability [2].

Large language models (LLMs) offer a promising alternative. Their ability to translate natural language into structured programs suggests a new interaction paradigm: rather than selecting from pre-built skills or training monolithic control policies, an LLM could generate executable robot code on demand, adapting to novel instructions in real time. However, most existing language-driven robotics systems restrict the model's role to high-level planning or bounded code composition, leaving open the question of whether LLMs can reliably synthesize low-level control logic under tight interface constraints [1, 4].

In this project, we investigate Code-as-Control, a system in which an LLM is responsible for generating full robotic control policies from natural language commands, using only a minimal API describing the robot's motion capabilities. The model is given no task-specific demonstrations, no skill library, and no environmental state beyond what is implicit in the prompt. The generated code is executed immediately on a Sawyer robotic arm in simulation, enabling rapid iteration and experimentation.

A key contribution of our work is the introduction of a natural-language policy editing loop. After observing a generated behavior, a user can provide feedback such as "make the circle twice as big" or "trace the path clockwise instead," and the LLM produces an edited version of the original policy. To our knowledge, this is one of the first systems to support iterative, language-driven editing of robotic control code itself, rather than editing symbolic plans or world models [3].

To evaluate this approach, we propose the Robotic Gestures Benchmark (RGB), a suite of gesture-based tasks designed to probe different dimensions of LLM-driven control generation. We compare Gemini-2.5 Pro and Gemini-3 Pro across multiple metrics, including task success, planning validity, code validity, control structure usage, and edit efficiency. Our results highlight both the promise of just-in-time policy generation and the rapid progress enabled by improvements in LLM capability.

II. Related Work

A. SayCan

SayCan ("Do As I Can, Not As I Say") introduced a framework in which an LLM interprets natural language instructions and selects from a predefined library of robotic skills, such as picking or placing objects [1]. An affordance model evaluates the feasibility of each proposed skill in the current environment, ensuring safe execution. While this approach demonstrated robust task execution and long-horizon planning, the LLM was not responsible for generating new behaviors; it merely selected and sequenced existing, human-authored skills. As a result, SayCan's expressiveness is fundamentally limited by the predefined skill set.

B. Code as Policies

The Code as Policies (CaP) framework advanced language-driven robotics by allowing LLMs to generate executable programs rather than selecting from fixed actions [4]. In CaP,

the model writes short Python programs that call a fixed set of high-level robot APIs, enabling the use of loops, conditionals, and compositional logic. This significantly increased flexibility compared to pure skill selection. However, the APIs exposed to the model remained coarse-grained, and novel behavior was constrained to recombinations of these predefined primitives.

C. RT-2

RT-2 adopts an end-to-end learning paradigm, training a single large model on internet-scale vision–language data and robot demonstrations [2]. Given an image and a text instruction, RT-2 directly outputs robot action tokens that can be executed without an intermediate planning or coding step. This enables impressive semantic generalization, such as identifying a pen as a tool for writing and manipulating it correctly. However, this approach requires massive datasets and computational resources, making it impractical for smaller research groups.

D. Natural Language World Modeling

Recent work from MIT and Cornell explores using natural language and LLMs to specify the "rules of the world," including action models and constraints [3]. While this enables flexible reasoning and symbolic planning, the LLM does not generate the robot's control policy itself. Instead, language is used to define abstractions that are later compiled into executable behaviors.

E. Positioning of This Work

Our project occupies a distinct point in this design space. Unlike SayCan, the LLM generates new code rather than selecting skills. Unlike CaP, the API exposed to the model is intentionally thin and low-level, allowing the model to invent novel trajectories rather than recombining high-level actions. Unlike RT-2, our approach relies on existing LLMs without retraining or large datasets. Finally, unlike prior work, we explicitly support iterative natural-language editing of robotic policies, enabling a human-in-the-loop refinement process.

III. System Overview

The system consists of five primary components: a command-line interface (CLI), an LLM planner, a minimal API, a safety layer, and an execution backend.

The CLI accepts a natural language command from the user and optionally includes a previously generated policy and feedback describing desired modifications. This input is forwarded to the LLM, which produces a Python function representing the robot's control policy.

The API exposes a minimal set of motion primitives for the Sawyer arm, including setting Cartesian targets, specifying joint configurations, planning collision-checked trajectories via MoveIt, executing plans, and controlling the gripper. Importantly, no higher-level task abstractions or helper functions are provided.

A safety layer enforces constraints on the generated code. An abstract syntax tree (AST) checker ensures that only approved language constructs and API calls are used, while a sandboxed execution environment enforces timeouts and memory limits. All motion plans are validated by MoveIt prior to execution.

All experiments in this work were conducted in simulation, allowing rapid iteration while avoiding hardware risk during development.

IV. Method: Code-as-Control

A. Prompt Design and Constraints

The LLM is provided with a system prompt of approximately 50 lines describing the Sawyer arm's kinematics and the available API functions. No examples, demonstrations, or task-specific hints are included. This intentionally low-context setup tests whether the model can infer appropriate control logic from minimal information.

B. Policy Generation

For each command, the LLM generates a Python function of the form `def run(api):`, where `api` provides access to the API. The function may include loops, conditionals, and timing logic, and is executed directly after passing safety checks.

C. Policy Editing Loop

After observing a policy execution, the user may request modifications using natural language. The editing prompt may include the original command, the previously generated code, and feedback describing the desired change; all fields are optional. The LLM then produces a revised policy, which is executed and evaluated in the same manner as the original. This loop enables incremental refinement without manual code editing.

V. Robotic Gestures Benchmark (RGB)

To evaluate the system, we introduce the Robotic Gestures Benchmark (RGB), consisting of five gesture-based tasks designed to probe different dimensions of LLM-driven policy generation. Each task is evaluated using a fixed natural-language prompt issued to the model, as listed below.

- **Multi-Step Gesture Sequence** – tests sequencing and temporal reasoning. *Prompt:* "Wave to me, then give me a high five, then return home."
- **Trajectory Generation** – tests continuous path synthesis. *Prompt:* "Draw a medium-sized circle in the air in front of you, then stop."
- **More Complex Path** – tests geometric reasoning over extended trajectories. *Prompt:* "Trace a figure-eight pattern in front of you."
- **Repetition and Timing** – tests loop construction and timing control. *Prompt:* "Wave to me five times quickly."
- **Code Edit Ability** – tests the model's ability to modify existing policies via language. *Prompt:* "Make the figure-eight much bigger."

VI. Experimental Evaluation

A. Setup

We evaluated two models: Gemini-2.5 Pro and Gemini-3 Pro. For each model, we performed five runs per task across five tasks, yielding 25 runs per model. All executions were observed by a human evaluator.

B. Metrics

- Task Success Rate: Binary success or failure based on observed execution.
- Planning Validity: Percentage of runs producing valid, collision-free trajectories.
- Code Validity: Percentage of generated policies executing without exception.
- Control Structure Usage: Whether appropriate loops or conditionals were used.
- Edit Efficiency: Whether edits succeeded and the number of follow-up prompts required.

C. Results

Gemini-2.5 Pro achieved an overall task success rate of 80%, with notable difficulties in precise trajectory control and geometric consistency. Planning and code validity were both approximately 60%, and policy edits often required multiple follow-up prompts.

Gemini-3 Pro achieved 100% task success, planning validity, and code validity across all runs. The model consistently used appropriate control structures and required only a single follow-up prompt for successful edits. Average latency for Gemini-3 was approximately five seconds for code generation.

VII. Discussion

The performance gap between Gemini-2.5 and Gemini-3 suggests that recent advances in LLM reasoning and code generation substantially improve reliability in robotic control contexts. Notably, Gemini-3 demonstrated robust geometric reasoning and produced more interpretable, structured policies.

The success of the editing loop highlights a promising interaction paradigm: rather than specifying complex behaviors upfront, users can iteratively shape robot behavior through natural language feedback. This mirrors modern software development workflows and supports rapid prototyping.

VIII. Limitations and Future Work

This work is limited to simulation-based evaluation and gesture-focused tasks. Future work will involve execution on physical hardware, introducing object interactions, and incorporating multimodal inputs such as vision. Additional safety mechanisms and robustness testing will be required for real-world deployment.

IX. Conclusion

We presented Code-as-Control, a system for just-in-time generation and editing of robotic control policies using large language models and a minimal API. Our results demonstrate that modern LLMs can reliably synthesize executable robot code and iteratively refine it via natural language. This approach offers a lightweight, flexible alternative to both skill-based systems and data-intensive end-to-end models, opening new possibilities for human-centered robot programming.

References

[1] Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil J Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, and Peter Pastor. Do as i can, not as i say: Grounding language in robotic affordances, 2022. URL https://arxiv.org/abs/2204.01691.

[2] Anthony Brohan, Noah Brown, Justice Carbajal, Yevgen Chebotar, Xi Chen, Krzysztof Choromanski, Tianli Ding, Danny Driess, Avinava Dubey, Chelsea Finn, Pete Florence, Chuyuan Fu, Montse Gonzalez Arenas, Keerthana Gopalakrishnan, Kehang Han, Karol Hausman, Alexander Herzog, Jasmine Hsu, Brian Ichter, Alex Irpan, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Isabel Leal, Lisa Lee, Tsang-Wei Edward Lee, Sergey Levine, Yao Lu, Henryk Michalewski, Igor Mordatch, Karl Pertsch, and Kanishka Rao. Rt-2: Vision-language-action models transfer web knowledge to robotic control, 2023. URL https://arxiv.org/abs/2307.15818.

[3] Aidan Curtis, Hao Tang, Thiago Veloso, Kevin Ellis, Tomas Lozano-Perez, and Leslie Pack Kaelbling. Llm-guided probabilistic program induction for pomdp model estimation. *OpenReview*, 2025.

[4] Jacky Liang, Wenlong Huang, Fei Xia, Peng Xu, Karol Hausman, Brian Ichter, Pete Florence, and Andy Zeng. Code as policies: Language model programs for embodied control, 2023. URL https://arxiv.org/abs/2209.07753.

X. Additional Resources

GitHub Repository:
https://github.com/AnishK05/code-as-control

Video Demonstrations (more available upon request):
https://drive.google.com/drive/folders/1uv5gBTqXsgAGQCbc6mNB4U61vvBUlyIE?usp=drive_link

XI. Acknowledgments

A special shout-out to Dr. Junhong Xu for letting me take on this endeavor and providing tremendous support and guidance over the course of the semester.