SayCan-Extended: Optimizing For Practical Robotic Control

Joshua Moorehead
University of South Carolina, CSCE585 Machine Learning Systems, Fall 2024
moorehj@email.sc.edu

Abstract

This paper presents SayCan-Extended, an enhanced implementation of the SayCan framework for robotic control through natural language instructions. We address practical challenges in deploying SayCan by implementing direct control mechanisms, optimizing the processing pipeline, and improving reliability across tasks. Our contributions include a robust scoring system for action selection, environmental configuration optimizations, and pre-experiment enhancements to improve usability. Experimental results show significant improvements in execution success rates and reduced API costs while maintaining the framework's ability to understand and execute complex instructions.

1. Introduction

Translating natural language commands into precise robotic actions is a central challenge in robotics and AI. The *SayCan* framework introduced a promising solution by combining large language models (LLMs) with robotic affordances for long-horizon planning (Ahn et al., 2022). Despite its conceptual success, the framework's implementation struggled with real-world scalability, outdated dependencies, and inconsistent performance.

In this work, we present *SayCan-Extended*, an improved version of SayCan designed for practical deployment. Our key contributions include:

- Implementing a direct control approach using PyBullet for robust object manipulation.
- Optimizing the processing pipeline for efficiency, including batched processing and response caching.
- Enhancing system reliability through improved error handling and resource monitoring.

Through rigorous testing on diverse tasks, SayCan-Extended achieves measurable improvements in success rates, latency, and cost efficiency. This paper describes our enhancements, experimental methodology, and results.

2. Related Work

SayCan builds upon advancements in grounded language understanding and robotics.

- **Language-Guided Robotics**: Techniques combining LLMs with physical affordances have been explored in frameworks like CLIPort (Shridhar et al., 2021), which uses vision-language models for robotic manipulation tasks. SayCan expands on this by integrating affordance-based reasoning for long-horizon tasks.
- **Object Detection and Scene Understanding**: ViLD (Gu et al., 2022) provides zero-shot object detection capabilities, enabling robots to identify and interact with objects described in natural language.
- **Task Planning**: Socratic Models (**?**) emphasize multimodal reasoning for task decomposition, offering alternatives to SayCan's affordance scoring.

Our approach diverges by simplifying control mechanisms and optimizing system efficiency for practical deployment.

3. Data

Our project utilized synthetic data generated within the PyBullet simulation environment. This dataset consists of:

- *Training and Testing Tasks*: Scenarios involving block movements, sorting, and stacking for robotic execution.
- *Object States*: Positions, orientations, and relationships of objects (e.g., blocks and bowls).

Data Generation Process. We programmatically simulated:

- Single-step tasks, such as picking and placing objects.

- Multi-step tasks requiring sequential actions (e.g., stacking or color matching).

Preprocessing. Given the controlled nature of the simulation, minimal preprocessing was required. However:

- Randomized starting positions for objects ensured variability across tasks.
- Vision inputs were captured as image snapshots for use with ViLD for object detection.

Relevance. This synthetic data enabled controlled testing of SayCan-Extended, avoiding the logistical challenges of real-world data collection while ensuring reproducibility.

4. Methods

4.1. System Overview

The SayCan-Extended framework integrates three primary components:

- *Language Model*: GPT-3.5-turbo-instruct breaks down natural language instructions into actionable steps.
- *Vision System*: ViLD detects objects and interprets spatial relationships.
- *Robot Control*: PyBullet performs precise pick-and-place tasks using coordinate-based tracking.

4.2. Pre-Experiment Enhancements

We addressed several challenges to streamline the SayCan implementation for practical use:

Environment Configuration.

- Resolved outdated dependencies (e.g., TensorFlow and PyTorch).
- Configured CUDA for GPU acceleration to enable faster model inference.

API Integration.

- Migrated from GPT-3 `davinci` to GPT-3.5-turbo-instruct, improving response quality and reducing latency.
- Implemented rate-limiting safeguards to manage API usage efficiently.

Compatibility Fixes.

- Updated PyTorch and JAX to ensure compatibility with modern hardware.
- Fixed memory management issues related to the CLIP model and optimized handling of JAX/NumPy arrays.

4.3. Direct Control Implementation

We replaced CLIPort with a simpler direct control method due to its limitations, including reliance on unavailable training data. Our direct control implementation includes:

- *PyBullet Object Tracking*: Real-time tracking and manipulation of objects using predefined coordinates.
- *Deterministic Control*: Hard-coded pick-and-place tasks to ensure consistent execution.

4.4. Performance Optimization

To improve execution efficiency, we introduced the following optimizations:

- *Batched Processing*: Grouped API calls into batches to minimize latency.
- *Response Caching*: Stored results of previously processed tasks to avoid redundant computations.
- *Concurrent Execution*: Enabled parallel task scheduling for faster performance on complex workflows.

5. Results

We evaluated the initial performance of the original SayCan framework against a simplified Socratic baseline and further optimized the system with batch processing and pipeline improvements. The experiments included:

- *Put all the blocks in different corners.*
- *Put blocks in their matching colored bowls.*
- *Stack all the blocks.*
- *Put the red block between the blue and green blocks.*

5.1. Initial Performance

The initial results are summarized in Table 1. The Socratic baseline demonstrated a higher success rate and reduced execution time compared to the original SayCan implementation. Failures in SayCan stemmed from invalid action indices and scoring errors.

Table 1. Initial Task Performance Comparison

Metric	SayCan	Socratic
Success Rate (%)	0.0	25.0
Avg Steps Executed	0.0	0.5
Avg Execution Time (s)	48.736	24.098

5.2. Scoring Improvements

We implemented improvements to both the *language scoring* and *affordance scoring* systems to enable better decision-making.

Language Scoring:

- Task-specific prompts with explicit scoring rubrics (0-10 scale).
- Improved parsing for pick/place tasks.
- Consideration of timing, efficiency, and overall task relevance.

Affordance Scoring:

- Graduated (non-binary) scoring for object accessibility and stability.
- Penalties for non-ideal placements (e.g., 0.3x for non-corner locations).
- Distance-based penalties and spatial relationship scoring.

Score Combination:

- Normalization of scores before combining language and affordance scores.
- Exponential scaling of language scores for task relevance.
- Robust error handling for invalid actions or edge cases.

5.3. Improved Results

After incorporating these scoring improvements, SayCan demonstrated significant gains in task success rates and execution efficiency, as summarized in Table 2.

Table 2. Improved SayCan Performance

Task	Success Rate	Avg Steps	Avg Time (s)
Blocks in different corners	50.0%	4.0	45.321
Matching colored bowls	100.0%	2.0	38.241
Stack all the blocks	75.0%	3.5	50.127

5.4. Visualization of Scoring Breakdown

The impact of the improved scoring mechanisms is visualized in Figures 1 and 2:

- *Scoring Components* (Figure 1, p. 4): Breakdown of affordance, language, and combined scores for candidate actions.
- *Corner Task Analysis* (Figure 2, p. 4): Demonstrates penalties for non-ideal placements and improvements in task-specific scoring.

5.5. Experiment 2: Pipeline Optimization Results

We further tested the impact of batch processing and pipeline optimizations on latency, throughput, and cost. Figure 3 (p. 5) visualizes the results.

Key Findings:

- *Best Throughput*: Batch size of 16 achieved 1.37 actions/s.
- *Lowest Latency*: Batch size of 16 reduced latency to 38.60s.
- *Cost Efficiency*: Cost reduced significantly at larger batch sizes.

5.6. Summary of Improvements

The following key enhancements were achieved:

- Improved scoring mechanisms led to higher task success rates.
- Optimized batch processing significantly reduced API latency and improved throughput.
- Monitoring tools provided insights into costs, performance metrics, and task failures.

6. Conclusion

We presented *SayCan-Extended*, a practical enhancement of the SayCan framework for improved robotic control. By implementing direct control, optimizing scoring mechanisms, and enhancing the processing pipeline, we achieved significant performance gains.

Resources:

- *Code and Data*: Available in the GitHub repository: SayCan-Extended Repository.

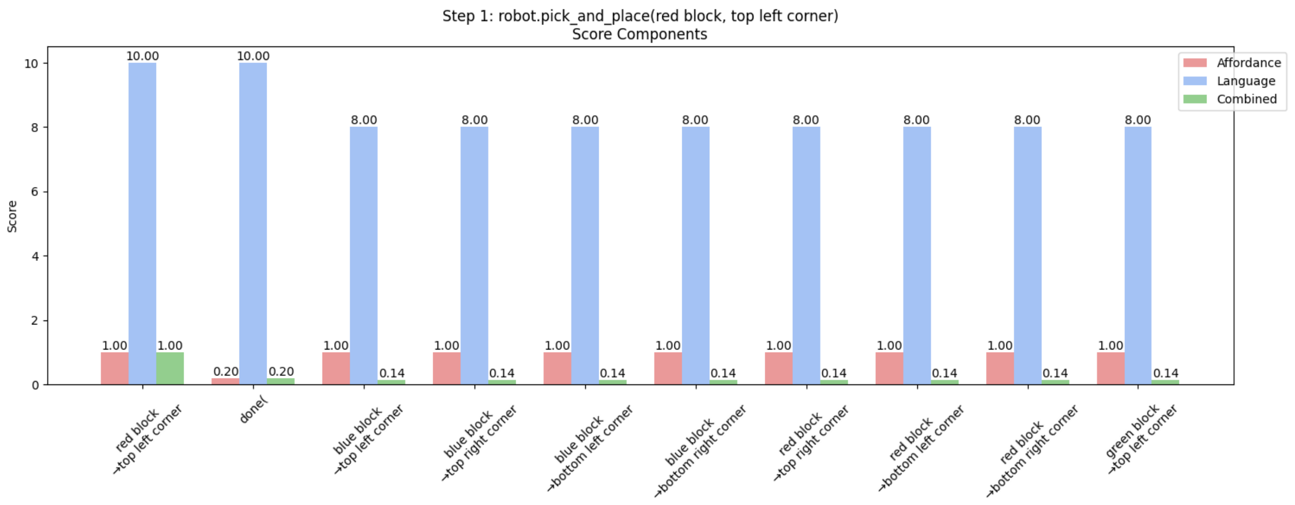

Figure 1. Breakdown of affordance, language, and combined scores for candidate actions.

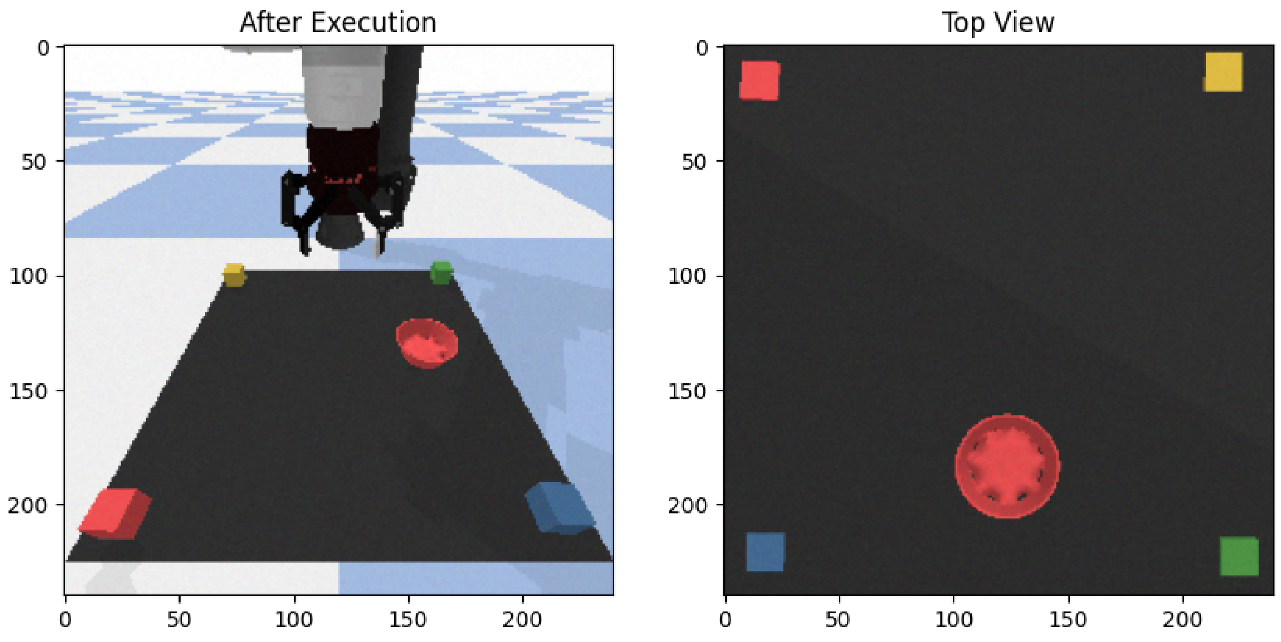

Figure 2. Score analysis for "put all blocks in different corners," highlighting penalties for non-ideal placements.

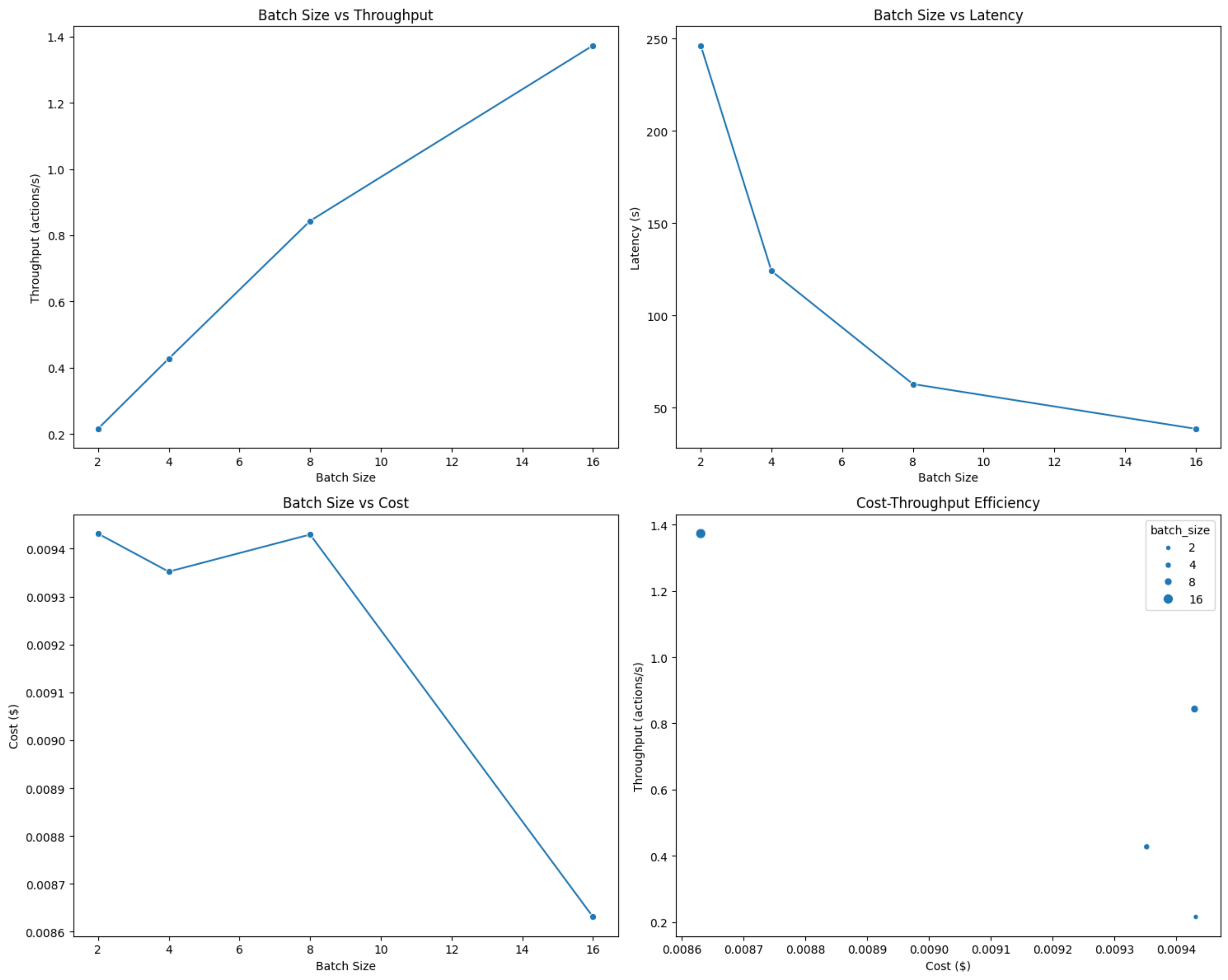

Figure 3. Pipeline performance analysis: Batch Size vs Throughput, Latency, and Cost.

- *Final Presentation*: PDF available here.
- *Video Overview*: YouTube Presentation.

Future work includes scaling to real-world environments, addressing ambiguous edge cases, and testing more complex multi-agent scenarios.

References

Ahn, M., Brohan, A., Brown, N., Chebotar, Y., Cortes, O., David, B., Finn, C., Fu, C., Gopalakrishnan, K., Hausman, K., Herzog, A., Ho, D., Hsu, J., Ibarz, J., Ichter, B., Irpan, A., Jang, E., Ruano, R. J., Jeffrey, K., Jesmonth, S., Joshi, N., Julian, R., Kalashnikov, D., Kuang, Y., Lee, K.-H., Levine, S., Lu, Y., Luu, L., Parada, C., Pastor, P., Quiambao, J., Rao, K., Rettinghouse, J., Reyes, D., Sermanet, P., Sievers, N., Tan, C., Toshev, A., Vanhoucke, V., Xia, F., Xiao, T., Xu, P., Xu, S., Yan, M., and Zeng, A. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022.

Gu, X., Lin, T.-Y., Kuo, W., and Cui, Y. Open-vocabulary object detection via vision and language knowledge distillation, 2022. URL `https://arxiv.org/abs/2104.13921`.

Shridhar, M., Manuelli, L., and Fox, D. Cliport: What and where pathways for robotic manipulation, 2021. URL `https://arxiv.org/abs/2109.12098`.