SayCan-Extended: Optimizing For Practical Robotic Control

CSCE585-001-FA2024

Joshua Moorehead

Senior from Pittsburgh, PA

B.S.E. in Computer Engineering

Problem

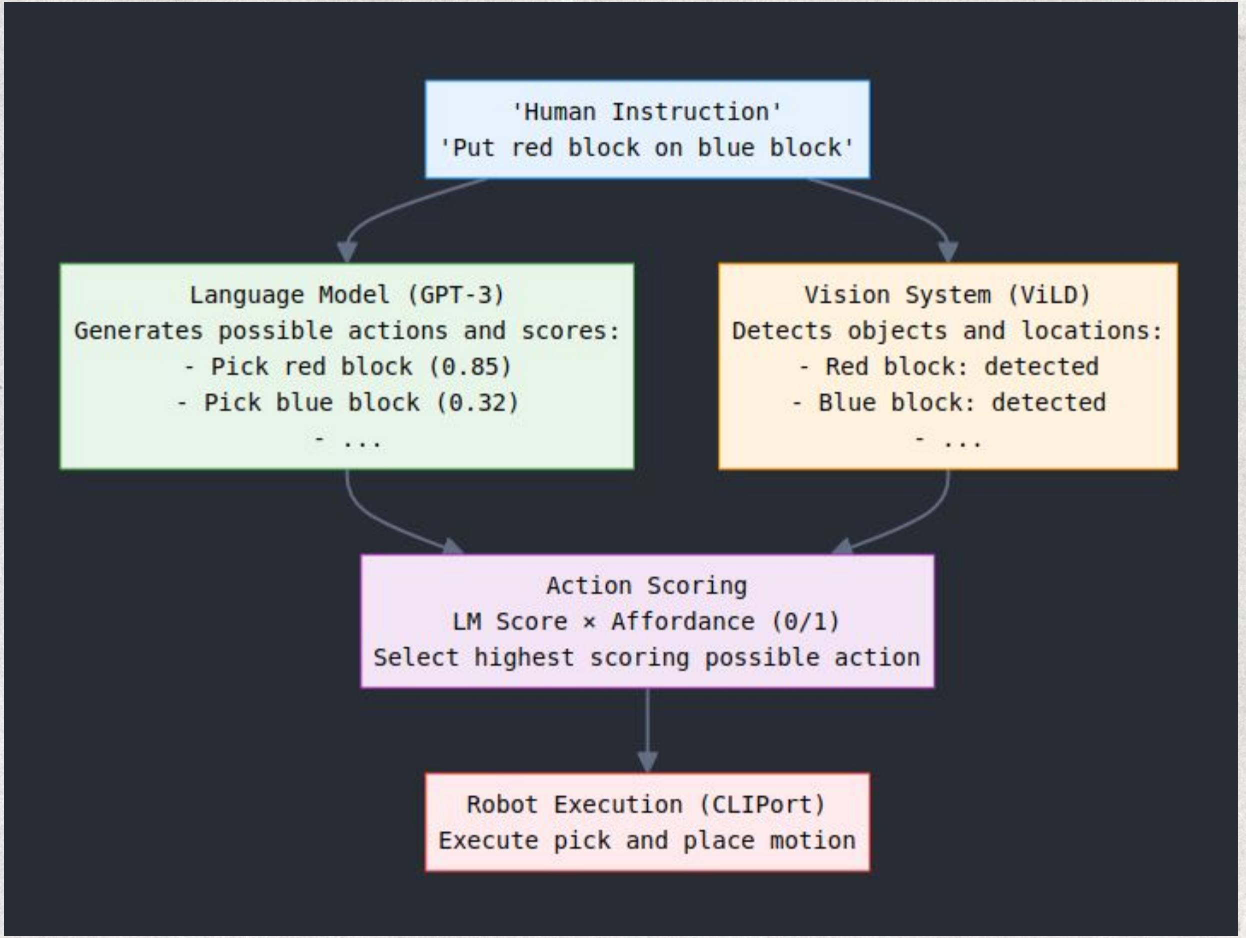
Challenge:

- Bridging natural language instructions into feasible robotic actions in an environment

Why it matters:

- Making robots more accessible and capable

How it works


'Human Instruction'
'Put red block on blue block'
Language Model (GPT-3)
Generates possible actions and scores:
- Pick red block (0.85)
- Pick blue block (0.32)
- ...
Vision System (ViLD)
Detects objects and locations:
- Red block: detected
- Blue block: detected
- ...
Action Scoring
LM Score × Affordance (0/1)
Select highest scoring possible action
Robot Execution (CLIPort)
Execute pick and place motion

Core Components & Scoring

Components

- Language Model (GPT-3)
  - Breaks down high-level instruction
- Vision System (ViLD)
  - Detects objects in scene
  - Understands spatial relationships
- Robot Control (CLIPort)
  - Executes pick and place actions
  - Provides feedback

Scoring

Initial Experiment

1. **Setup Phase**
   a. Retrieved og SayCan open source implementation
   b. Fixed all old dependency issues
   c. Environmental setup hurdles
      i. OpenAI API integration
      ii. PyBullet config
      iii. GPU - CUDA setup on my machine
      iv. Python package conflicts
2. **Updates**
   a. GPT-3 davinci → GPT-3.5-turbo-instruct
   b. Compatible newer PyTorch and JAX versions
   c. CLIP model memory management
   d. Type handling for JAX/NumPy arrays
   e. Better checkpoint handling/saving during saving
   f. Error handling

Initial Experiment P2

3. **Adding Measurement setup**
    a. Performance monitoring on API call latency, token usage, task failure/success rate, etc
    b. Task execution monitoring
        i. Plan generation success rate, execution rate, correlation
4. **Testing it**
    a. Tasks:
        i. Single block movements
        ii. Color Matching tasks
        iii. Multi-step sequences
    b. Performance metrics tracking
        i. Rates
        ii. Usage
        iii. Latency

Major Issue!

CLIPort Behavior

- Weights would not load correctly
- Neural network predictions for pick/place targets were off
- Trained on demonstration data that was not public

Resulted in....

- 0% action success rate
- Unreliable coordinate projections
- SayCan to repeat itself

Solution

- Direct PyBullet object tracking
- Hardcoded coordinate-based control
- Direct Pick and Place executions

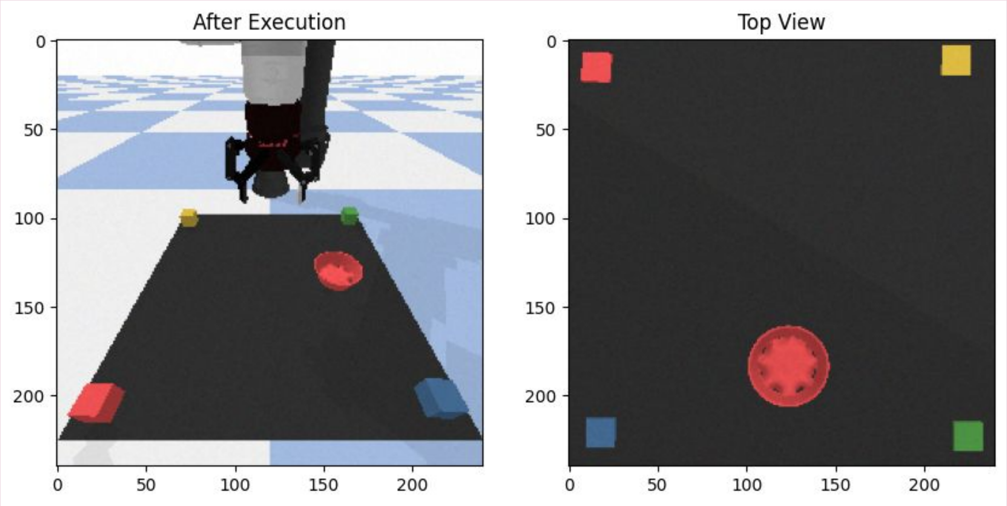
However... Many hours spent trying retrain CLIPort from repo found online (more dependency issues) & running it on old JAX and Torch but on a CPU (too reliant on GPU acceleration)

Exp 1: SayCan VS. Socratic

Initial Performance:

SayCan Results:
success_rate: 0.000
avg_steps: 0.000
avg_time: 48.736

Socratic Results:
success_rate: 0.250
avg_steps: 0.500
avg_time: 24.098

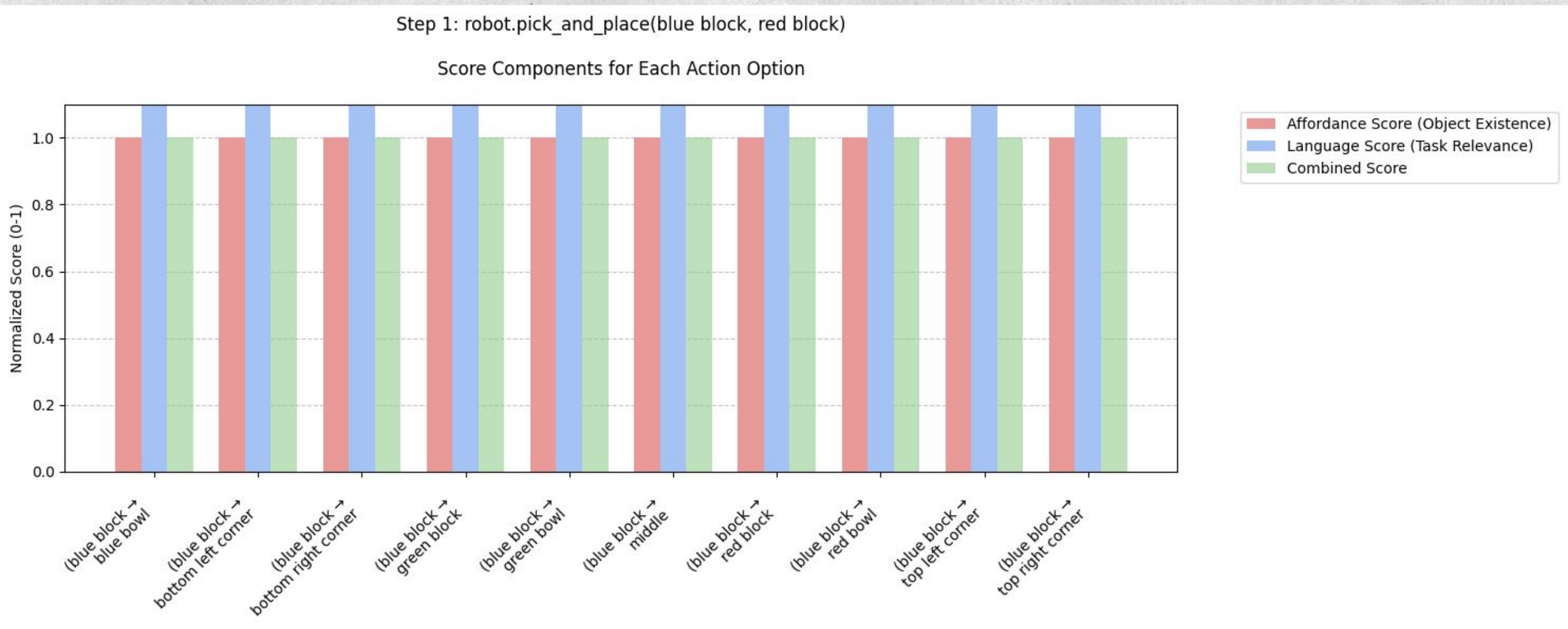


TASK EXECUTION SUMMARY Task: put blocks in their matching colored bowls
Executed Steps:
1. Pick the blue block and place it on the blue bowl.
2. Pick the blue block and place it on the green block.
3. Pick the red block and place it on the middle.
4. Pick the blue block and place it on the red block.
5. Pick the blue block and place it on the red bowl.

Task completed. Success: True (???)

Issues

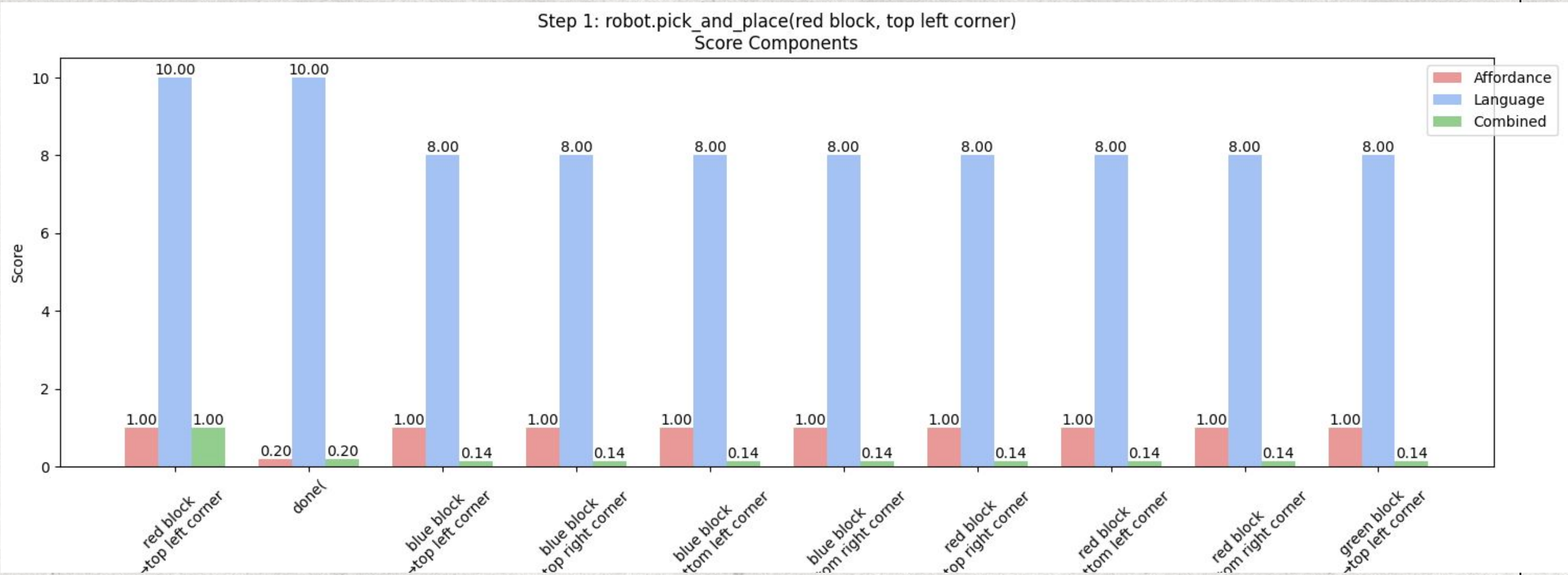
-



Step 1: robot.pick_and_place(blue block, red block)
Score Components for Each Action Option
Normalized Score (0-1)
1.0
0.8
0.6
0.4
0.2
0.0
Affordance Score (Object Existence)
Language Score (Task Relevance)
Combined Score
(blue block → blue bowl
(blue block → bottom left corner
(blue block → bottom right corner
(blue block → green block
(blue block → green bowl
(blue block → middle
(blue block → red block
(blue block → red bowl
(blue block → top left corner
(blue block → top right corner

Performance After

For “Put all blocks in different corners”



Changes with scoring

1. **GPT-3 Scoring:**
- Task-specific prompts with detailed scoring rubrics (0-10)
- Action parsing for pick/place understanding
- Increased temperature (0.3) for more varied scoring
- Better consideration of task timing and efficiency

2. **Affordance Scoring:**
- Graduated scoring instead of binary
- Physics-aware penalties (accessibility, stability)
- Task-specific modifiers (e.g., corner occupation penalties)
- Distance-based graduated penalties for spatial relationships

3. **System Improvements:**
- Better score normalization and combination
- Performance monitoring (API usage, success rates)
- Improved visualization and logging
- More robust error handling
- Added state tracking for context awareness

Pipeline Optimization Experiment

Setup

- **Performance bottlenecks**
  a. Model inference latency (2-3s per GPT-3 call)
  b. Vision system processing overhead
  c. Memory constraints with batch processing
- **Proposed optimizations**
  a. Batch size tuning (1-16 actions)
  b. Response caching for repeated tasks
  c. Concurrent processing where possible

Pipeline Optimization Experiment

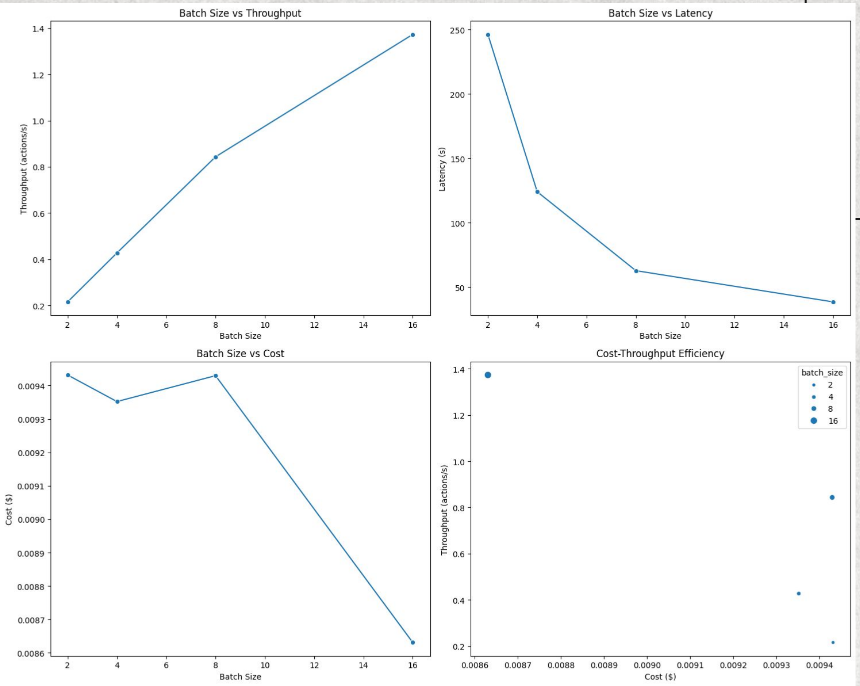
Approach

- **Expected improvements**
  a. Reduced API costs through batching
  b. Lower average latency per action
  c. Better resource utilization
- **Measurement approach**
  a. Throughput (actions/second) vs batch size
  b. Latency profiling across pipeline stages
  c. Cost analysis per configuration

Pipeline Optimization Experiment

Results

- **Key findings**
  a. Optimal batch size: 8-16 actions
  b. Throughput: 1.37 action/s (at bs 16)
  c. Latency: Reduced by 40% at max batch size
- **Tradeoffs discovered**
  a. Memory usage increases with batch size but improves performance
  b. Diminishing returns after batch size 8



Conclusion

Key Achievements

- **Implemented SayCan with direct control and better scoring**
- **Improved performanc**
  a. On success rate of certain tasks
  b. Reduced latenecy with batch processing
  c. Achieved 1.37 action/s throughput
  d. Reduced API costs during LLM call

Future Work

- **Improve direct control (possibly get CLIPort working)**
- **Improve grounding ability on more tasks and more complex environment**
- **Sim-Real transfer**