LLM Evolutionary Policy Search vs. Reinforcement Learning for Underspecified Tasks

Allen Jue (mrallenjue@utexas.edu)

***Abstract*—We compare LLM-based evolutionary policy search against model-free reinforcement learning for latent rule discovery in robotic manipulation. Given only reward signals and no task specification, both agents must infer an underlying organizational rule from environment feedback alone. We employ ShinkaEvolve, an evolutionary code-search loop inspired by AlphaEvolve, which iteratively proposes and refines executable Python policies, and compare it against a PPO baseline with a shared per-block encoder across twelve task variants spanning positional pickup, color sorting, and table-setting.**

We evaluate both agents along three axes: sample efficiency, generalization, and interpretability. While PPO achieves near 100% success across all tasks, ShinkaEvolve attains 3–47$\times$ higher sample efficiency on selected color and positional tasks and produces interpretable policies that generalize beyond the training distribution. However, on table-setting tasks, ShinkaEvolve fails to capture semantic affordances and often converges to cyclic or incorrect arrangements. These results indicate that LLM-based evolutionary search is effective when latent rules align with pretrained semantic priors, but produces brittle policies when tasks require grounded physical reasoning.

I. Introduction

A central challenge in robot manipulation is generalizing to novel organizational tasks specified only by reward signals. Human experts can infer such tasks almost instantly ("sort by color," "pick the second from the right") because they bring rich semantic priors about spatial relations, object categories, and common conventions. These priors allow humans to rapidly interpret sparse feedback and infer the latent rule governing a task.

However, many real-world manipulation tasks are inherently underspecified. Multiple reasonable solutions may exist, and the correct behavior may depend on latent user preferences rather than universally shared conventions. For example, organizing a room or setting a table can be done in several valid ways. While semantic knowledge can suggest a plausible arrangement, a particular user may prefer a different configuration. In such cases, an agent must adapt its behavior based on feedback signals rather than relying solely on prior knowledge.

The recent emergence of multimodal large language models (LLMs) trained on internet-scale data raises a natural question: can the semantic understanding encoded in LLMs substitute for the thousands of environment interactions that traditional RL techniques require? Ahn et al. [1] has shown that LLMs can plan over robot primitives and Huang et al. [5] demonstrates reasoning over closed-loop feedback. More recently, Chen et al. [3] utilize LLMs for warm-start RL training by generating imperfect rule-based controllers that accelerate policy learning. These approaches demonstrate that LLMs can guide or accelerate policy learning when the task specification is known.

However, direct comparisons between LLM-based agents and tabula-rasa RL baselines on rule-discovery tasks remain sparse. In particular, it is unclear whether semantic priors help agents infer latent rules that are not explicitly specified but must instead be discovered from reward feedback.

We design an experiment to probe this gap directly. We utilize *ShinkaEvolve*, an evolutionary code-search loop inspired by AlphaEvolve, that iteratively proposes and refines executable Python policies against a reward-based fitness function without ever receiving the rule specification. We compare ShinkaEvolve against a PPO baseline with a shared per-block encoder across twelve task variants in a MuJoCo/Robosuite tabletop environment, and evaluate both agents along three axes:

- **Sample efficiency**: episodes required to reach convergence, measured against a shared environment step budget.
- **Generalization**: transfer to block counts $N \in [3, 8]$ without retraining, after training on $N \in [3, 6]$.
- **Interpretability**: whether the discovered policy reflects the latent rule in human-readable form.

We additionally vary reward density—dense (reward after every action) vs. sparse (reward at episode end)—to assess robustness to feedback sparsity.

Our results reveal a fundamental asymmetry. ShinkaEvolve achieves 3–47$\times$ greater sample efficiency than PPO on color-sorting and positional pickup tasks, produces interpretable policies that generalize to unseen block counts, and degrades gracefully under sparse reward. On table-setting tasks, PPO succeeds on table-setting by exhaustively exploring action combinations until saturation, but ShinkaEvolve surprisingly fails to utilize semantic priors to organize the table.

II. Related Work

A. LLMs for Robotic Planning and Control

Recent work has demonstrated that large language models (LLMs) encode semantic priors that can be leveraged for robotic planning and control. Ahn et al. [1] show that LLMs can decompose high-level instructions into sequences of robot actions by grounding language in learned affordance models. Similarly, Huang et al. [5] introduce an "inner monologue" framework in which LLMs incorporate environment feedback into iterative reasoning, enabling improved performance in closed-loop manipulation tasks.

These approaches demonstrate that LLMs can guide behavior when task specifications are provided explicitly or through natural language. However, they do not address settings in which the task must be inferred solely from reward signals.

B. Iterative Reasoning and Program Synthesis with LLMs

A parallel line of work has explored improving LLM reasoning through structured prompting and iterative refinement. Chain-of-thought prompting [14] elicits intermediate reasoning steps that improve performance on complex tasks. Self-Refine [8] extends this idea by enabling models to iteratively critique and refine their own outputs, while multi-agent debate frameworks [4, 7] leverage multiple interacting agents to improve reasoning quality.

More recently, program synthesis and evolutionary approaches have emerged as a powerful paradigm for leveraging LLMs. Novikov et al. [10] introduce AlphaEvolve, a system that uses LLMs to iteratively generate and refine code for scientific discovery. Building on this, Lange et al. [6] propose ShinkaEvolve, which frames code generation as an open-ended evolutionary process guided by execution-based feedback.

These methods demonstrate that LLMs can search over program spaces using feedback signals, but they have primarily been applied to static algorithmic problems rather than embodied robotic tasks.

C. Reinforcement Learning for Robotics

Reinforcement learning (RL) provides a general framework for learning control policies from reward signals by modeling interaction as a Markov Decision Process [2]. Deep RL has achieved strong results in robotic manipulation, locomotion, and navigation [13]. However, it typically suffers from high sample complexity and requires large numbers of environment interactions.

This limitation is particularly evident in manipulation tasks. Rajeswaran et al. [11] show that learning dexterous behaviors from scratch often requires extensive data or demonstrations. Proximal Policy Optimization (PPO) [12] is a widely used baseline due to its stability and performance, but it remains fundamentally reliant on trial-and-error exploration.

Recent work has explored combining LLMs with RL to improve sample efficiency. For example, Chen et al. [3] use LLM-generated policies to warm-start RL training, reducing the number of required interactions. However, these approaches assume that the task structure is at least partially known.

D. Summary and Gap

Prior work establishes that LLMs can encode useful semantic priors, perform iterative reasoning, and generate executable programs, while RL provides a general mechanism for learning from reward. However, direct comparisons between LLM-based policy search and model-free RL on *latent rule discovery* tasks remain limited.

In particular, it is unclear whether LLM-based evolutionary methods can match or exceed RL in settings where the task is not specified and must be inferred purely from reward feedback. This work addresses that gap by evaluating an LLM-driven evolutionary framework against a PPO baseline across a suite of underspecified manipulation tasks.

III. Data

A. Task Distribution

We evaluate our methods on a suite of twelve manipulation tasks spanning positional pickup, color sorting, and table-setting scenarios inspired by household manipulation benchmarks in RoboCasa [9]. Each task is defined by a latent rule that determines the correct behavior but is never explicitly provided to the agent. Instead, agents must infer the rule purely from reward feedback.

For each episode, a set of N blocks is generated with $N \in [3, 6]$ during training. For pickup tasks, blocks are assigned colors uniformly at random from a fixed palette of six categories $\{red, blue, yellow, green, magenta, cyan\}$, and their initial ordering is randomized. For color sorting tasks, blocks are assigned colors from $\{red, blue\}$.

We consider three rule families: (i) extremal positional rules (e.g., leftmost or rightmost selection), (ii) relational positional rules (e.g., k-th from left or right), and (iii) color-conditioned sorting rules (e.g., grouping blocks by color).

B. Simulation Environments

All training and quantitative evaluation are conducted in a lightweight symbolic environment, which represents the scene as a structured state consisting of block indices, positions, and color labels. Transitions are computed analytically without physics simulation, enabling high-throughput experimentation.

For qualitative validation and realistic execution, we deploy policies in physics-based simulation environments using *robosuite* [15] and *RoboCasa* [9]. Robosuite provides a modular MuJoCo-based framework for robot manipulation tasks, including pick-and-place primitives with a Franka Panda arm. RoboCasa extends this with large-scale, semantically rich household environments, including table-setting scenarios and object affordances.

In these environments, policies operate through predefined pick-and-place primitives, executing multi-step motion sequences under realistic dynamics. While our primary quantitative results are obtained in the symbolic setting, these simulations verify that learned policies can be executed in physically grounded environments.

IV. Methods

A. Alternative LLM-Based Approaches

Before adopting evolutionary program search, we explored several alternative LLM-based formulations for latent rule discovery. We first considered a structured hypothesis-selection pipeline, in which the LLM generates multiple candidate rules (e.g., "leftmost selection," "color grouping," "k-th from right") and selects a single most likely hypothesis to be compiled into a policy. Although this formulation incorporates reward feedback through trajectory summaries, it introduces a

bottleneck in which a single discrete rule must be selected prior to execution.

Qualitatively, this approach was unstable under sparse reward conditions and only generated working policies for extremal pickup tasks (Appendix A). The model frequently committed to incorrect hypotheses, and subsequent reward feedback was insufficient to reliably correct these errors. This led to premature convergence to suboptimal policies.

We also explored ReAct-style prompting, where the model alternates between reasoning over trajectory summaries and proposing actions, as well as iterative hypothesis refinement, where candidate rules are updated based on reward signals. These methods exhibited similar limitations, including unstable reasoning trajectories and poor robustness to noisy or sparse feedback.

B. ShinkaEvolve: Evolutionary Program Search

ShinkaEvolve formulates policy learning as an evolutionary search problem over executable programs. Instead of committing to a single latent rule, the system directly generates candidate policies, executes them in the environment, and selects high-performing programs based on empirical reward.

Each iteration consists of four steps: (i) evaluation of candidate policies on environment rollouts, (ii) summarization of successful and failed trajectories, (iii) LLM-based generation of improved policy variants conditioned on prior experience, and (iv) selection of policies based on measured performance.

This formulation removes the need for explicit rule identification and instead optimizes directly over behavioral outcomes. Moreover, rather than reimplementing complex hypotheses generating pipelines, we defaulted to a well-refined evolution-guided algorithm.

Each ShinkaEvolve experiment consists of $R = 3$ independent runs. Each run begins with an initial seed program and proceeds for $G = 20$ generations. At each generation, the LLM proposes $K = 5$ candidate programs, each evaluated using $E = 10$ episodes.

The total number of environment episodes per full experiment is therefore:

$$N_{\text{Shinka}} = R \times (1 + G \cdot K) \times E = 3030.$$

The term 1 accounts for the evaluation of the initial seed program prior to evolutionary updates.

C. Reinforcement Learning Baseline

We implement Proximal Policy Optimization (PPO) as a standard model-free baseline. The policy uses a shared per-block encoder that processes object-level features such as position and color. These embeddings are aggregated into a global representation used by policy and value heads.

PPO is trained for up to 30,000 episodes using standard hyperparameters, including learning rate 3×10^{-4}, discount factor $\gamma = 0.99$, GAE parameter $\lambda = 0.95$, and clipping parameter 0.2. Evaluation is performed every 100 episodes on a deterministic validation set of 100 episodes, with early stopping when perfect success is achieved.

D. Primitives and Robosuite Environments

a) Abstract Environment.: During training, we use a lightweight symbolic simulator that represents a tabletop as a list of N block states, each storing a row index, position (x, y), and color. There is no physics: actions are discrete block selections, transitions update block states analytically, and reward is computed directly from the resulting configuration. This decoupling from the physics engine enables thousands of episodes per minute.

b) Robosuite Environment.: For the final LLM vs. RL comparison and visual experiments, we use a MuJoCo/Robosuite simulation with a Panda arm executing real pick-and-place motions. Two primitives underlie all manipulation:

– `pick_up_block(env, i)` runs a four-phase state machine (approach, lower, grasp, lift) and returns a success flag and step count.

– `put_down_at_position(env, x, y)` places the currently held block at target coordinates (x, y) via a symmetric lower, release, and retract sequence.

Pickup episodes call `pick_up_block` once; sort episodes interleave three calls to each primitive per swap. Empirically, these primitives are robust and succeed for 1-8 blocks.

All episodes across all rounds count toward the shared budget K. Policy code is sandboxed with a strict API (primitives only).

V. Experiments

A. Primitive Reliability

Across 200 repeated pick-and-place cycles in Robosuite, `pick_up_block` and `put_down_at_position` succeed reliably. Failure modes at the policy level therefore reflect decision errors, not primitive unreliability.

B. Pickup Tasks

Pickup tasks are dominated by extremal and relational index reasoning (e.g., selecting the leftmost, rightmost, or k-th block). In this regime, ShinkaEvolve consistently requires fewer environment interactions to reach optimal performance.

For extremal selection tasks (leftmost and rightmost), both methods achieve perfect success rates, but ShinkaEvolve converges approximately $3.8\times$ faster. The improvement is more pronounced in relational tasks, where PPO requires significantly more samples to infer positional structure.

A key exception is the "2nd left" task, where ShinkaEvolve fails entirely despite successful PPO convergence. This failure mode is attributable to over-commitment to a heuristic favoring high-index selections, which does not generalize symmetrically across left-right transformations.

C. Color Sorting Tasks

Color sorting tasks require global reasoning over object sets rather than local selection. These tasks expose a clearer separation between dense and sparse reward regimes.

Under dense reward, PPO performs strongly but requires up to $30{,}000$ episodes to reach convergence on sorting red blocks to the left. ShinkaEvolve achieves comparable or higher

TABLE I
PPO VS SHINKAEVOLVE ACROSS TASK FAMILIES. DENSE DENOTES STEP-WISE REWARD; SPARSE DENOTES END-OF-EPISODE REWARD. BOLD INDICATES BEST SAMPLE EFFICIENCY.

Variant	PPO SR	Shinka SR	N_{PPO}	N_{Sh}	$N_{\text{PPO}}/N_{\text{Sh}}$	SR Ratio
Pickup Tasks						
rightmost	1.0000	1.0000	700	180	**3.89**	1.0000
2nd right	1.0000	1.0000	5800	1530	**3.79**	1.0000
leftmost	1.0000	1.0000	700	180	**3.89**	1.0000
2nd left	1.0000	0.0000	1800	3030	0.59	0.0000
Color Sorting Tasks						
reds left (dense)	0.9860	1.0000	30000	630	**47.62**	**1.0142**
reds left (sparse)	0.9970	1.0000	1800	180	**10.00**	**1.0030**
blues left (dense)	0.9560	0.6267	15500	3030	**5.12**	0.6555
blues left (sparse)	0.9320	1.0000	20800	1380	**15.07**	**1.0730**
Table Manipulation Tasks						
table-setting (dense)	1.0000	0.4200	700	3030	0.23	0.4200
table-setting (sparse)	1.0000	0.0000	8400	3030	2.77	0.0000

success rates with significantly fewer samples on the easiest configurations, particularly the "reds left" setting, where it achieves up to $47\times$ improvement in sample efficiency.

However, this advantage is not uniform across reward regimes. In several cases, PPO benefits more from dense intermediate feedback, while ShinkaEvolve exhibits higher variance under dense reward due to sensitivity in program-level evaluation signals. In contrast, under sparse reward, ShinkaEvolve becomes more stable on certain tasks (e.g., "reds left"), suggesting that sparse feedback can be beneficial in the evolutionary search process. This may be due to increased input complexity degrading LLM response quality.

Overall, PPO remains more consistent across both reward regimes, while ShinkaEvolve exhibits stronger but less uniform gains that depend on alignment between reward structure and program heuristic search.

D. Table Manipulation Tasks

Table manipulation tasks are the most challenging regime, requiring both spatial reasoning and implicit affordance modeling. Unlike pickup and sorting tasks, these environments words like `fork` and `knife` that may be used in conjunction with index-based heuristics.

In this setting, PPO consistently achieves high success rates under both sparse and dense feedback regimes, indicating that gradient-based learning can eventually try every permutation of table-setting arrangements given sufficient exploration.

In contrast, ShinkaEvolve performs significantly worse, particularly under sparse reward, where it fails to discover stable policies. Even under full feedback, performance remains substantially below PPO, with success rates dropping to 0.42 in the best case.

Failure analysis suggests that evolved programs tend to overfit to shallow structural patterns (e.g., repeated action sequences or cyclic swaps) without capturing affordance constraints required for stable manipulation.

This result is notable given that large language models are trained on extensive internet-scale data containing rich semantic descriptions of everyday tasks such as table setting. One might therefore expect that such priors would transfer to structured manipulation policies (e.g., placing utensils and plates in conventional spatial configurations when presented with corresponding objects).

E. Sample Efficiency

Across pickup and color sorting tasks, ShinkaEvolve achieves between $3\times$ and $47\times$ improvements in sample efficiency over PPO. These gains are most pronounced in tasks with clear combinatorial or rule-based structure, where program synthesis can directly represent the underlying decision boundary.

However, this advantage does not extend to table manipulation tasks, where PPO remains significantly more reliable. This indicates that LLM-driven program evolution is most effective when task structure aligns with discrete symbolic reasoning, but less effective when success depends on continuous physical interaction modeling.

VI. INTERPRETABILITY OF LEARNED POLICIES

Beyond performance and sample efficiency, we additionally analyze the interpretability of the policies produced by ShinkaEvolve. Interpretability is defined here as the degree to which a learned policy can be directly understood as a human-readable procedure that reflects the underlying task structure.

Unlike PPO, which encodes behavior in distributed neural representations that are not directly inspectable, ShinkaEvolve produces explicit Python programs. This allows each policy to be directly examined as a symbolic object, enabling qualitative inspection of decision rules, control flow, and implicit heuristics.

In practice, we observe that successful evolved policies often recover simple and structured rules that correspond to intuitive task decompositions. For example, in color-sorting tasks, the model frequently discovers greedy swap-based procedures that iteratively move target-colored blocks toward a desired region (VI). These programs are compositional and locally interpretable, typically consisting of explicit selection rules over indices and deterministic swap operations.

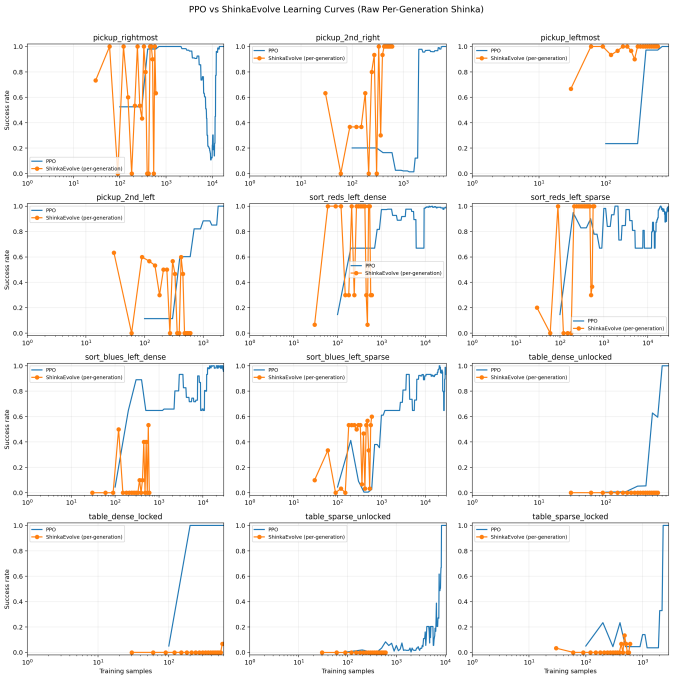

Fig. 1. (a) Raw performance over time

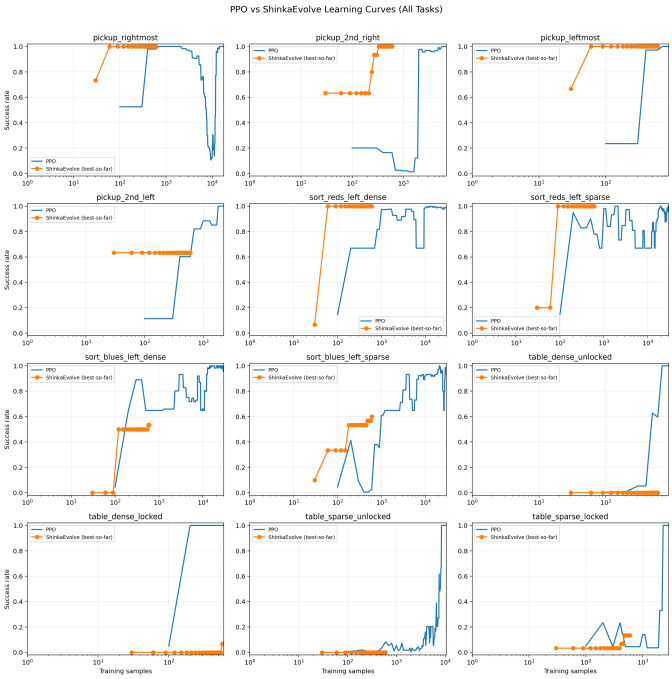

Fig. 2. (b) Best policy over time

Fig. 3. Learning curves for ShinkaEvolve across tasks. The raw curve (left) shows high variance and instability in candidate policies, while the best-so-far curve (right) highlights periods of rapid improvement followed by stagnation, indicating premature convergence in the evolutionary process.

Listing 1. We present a representative policy generated by ShinkaEvolve after 20 generations on the color-sorting task. This policy demonstrates the emergence of a simple but interpretable heuristic for clustering red blocks to the left. This policy was not explicitly programmed but emerged through reward-driven evolutionary refinement. It encodes a greedy swap heuristic that approximately implements the latent rule of grouping red blocks to the left.

```
def policy(state, memory):
    """Improved heuristic policy to cluster red
    blocks to the left."""
    n = int(state.get("num_blocks", 0))
    max_blocks = int(state.get("max_blocks", 10))
    noop_action = int(state.get("noop_action",
    max_blocks * max_blocks))
    colors = state.get("colors", [])
    if n < 2 or len(colors) < n:
        return noop_action

    red = "red"
    left_idx = None
    right_idx = None

    for i in range(n):
        if colors[i] != red:
            left_idx = i
            break

    for j in range(n - 1, -1, -1):
        if colors[j] == red:
            right_idx = j
            break

    if left_idx is None or right_idx is None or
    left_idx >= right_idx:
        return noop_action

    return left_idx * max_blocks + right_idx
```

However, interpretability does not imply correctness or completeness. Many evolved policies implement heuristics that are only locally valid and fail to capture global invariants of the task. This results in policies that are readable but brittle, particularly under distribution shifts such as changes in block count, color frequency, or spatial ordering.

A. Failure Modes

We identify three dominant failure modes:

a) PPO: Slow rule induction: PPO requires a large number of environment interactions to infer relational structure, particularly for tasks such as k-th element selection that depend on non-local reasoning. While PPO improves steadily with additional training, it converges slowly and may require substantial training to reach perfect performance within a fixed budget. For color sorting, it may plateau just below perfect performance.

b) ShinkaEvolve: Heuristic collapse and limited exploration: ShinkaEvolve is prone to prematurely converging on partially correct heuristics that achieve high reward but fail under distribution shifts (e.g., left-right symmetry or varying block counts). Once such heuristics dominate the population, further evolution may stagnate, limiting exploration of alternative program structures. This stagnation is evident in the learning curves (Fig. 3), where performance plateaus after early gains and exhibits high variance across generations. Encouraging diverse and meaningful exploration in program space remains a challenge, particularly when feedback signals are sparse or noisy.

c) Table manipulation: Grounding Limitations: PPO can solve the table-setting task through continued exploration, whereas ShinkaEvolve's performance depends heavily on prompt design, model capability, and the quality of textual feedback, leading to high variability across runs. These results highlight a gap between semantic reasoning and executable, affordance-aware control.

VII. Limitations

While the results highlight clear differences between LLM-based evolutionary search and PPO, several limitations affect the strength and generality of the conclusions.

First, we do not control for LLM stochasticity. We did not fix a random seed or temperature for the LLM (ChatGPT 4.1 mini), which introduces variance across runs and limits reproducibility. As a result, some performance differences may reflect sampling noise rather than systematic advantages of the evolutionary approach.

Second, our experiments rely on a closed-source model. This limits control over decoding strategies, calibration, and architectural details, and prevents controlled ablations. It also makes it difficult to determine whether observed behaviors such as heuristic collapse or instability are inherent to the method or specific to the model.

Third, we did not evaluate open-weight LLMs due to time constraints. Testing local models (e.g., Gemma-family models) would allow more controlled experimentation and could reveal whether smaller or differently trained models exhibit similar evolutionary dynamics.

Fourth, the policy representation is restricted to executable Python programs over symbolic state. While this enables interpretability, it constrains the hypothesis space and biases the search toward brittle, discrete heuristics rather than more robust or continuous strategies.

Fifth, the reward structure is relatively simple and fully observable in the symbolic environment. Real-world manipulation often involves delayed, noisy, or partial rewards, which may further challenge both PPO and evolutionary approaches.

Finally, an alternative formulation would be to directly prompt the LLM to output a target configuration (e.g., block placements as JSON). While this may work for fixed-size problems, it does not scale well to variable numbers of objects and does not naturally produce reusable policies over arbitrary N. This motivates our focus on programmatic policies, but the tradeoff remains worth exploring.

VIII. Conclusion and Future Work

We study latent rule discovery in robotic manipulation under reward-only supervision, comparing LLM-based evolutionary program search (ShinkaEvolve) with model-free reinforcement learning (PPO).

The results show a clear tradeoff. ShinkaEvolve achieves substantial gains in sample efficiency, up to $47\times$, on tasks with strong combinatorial or symbolic structure such as positional selection and color sorting. It also produces interpretable policies that generalize across different numbers of objects. However, these advantages are not consistent across all task types. In table-setting tasks that require implicit affordance reasoning, PPO is significantly more reliable, while ShinkaEvolve exhibits instability, premature convergence, and poor grounding.

These findings suggest that LLM-driven evolutionary search is effective when the task aligns with discrete symbolic reasoning, but less effective when success depends on physical constraints or unobserved structure.

Future work should focus on improving reproducibility by controlling LLM sampling and running multiple trials, as well as evaluating open-weight models to better understand the role of model architecture and training data. Hybrid approaches that combine evolutionary program search with gradient-based learning may help bridge the gap between symbolic reasoning and continuous control. Improving exploration in program space, for example through diversity-promoting mechanisms, could mitigate heuristic collapse. Finally, incorporating richer representations and stronger grounding in object affordances, such as multi-modal LLMs along with evaluation on real robotic systems, would be necessary steps toward deploying these methods in practical settings.

Acknowledgments

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, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022. URL https://arxiv.org/abs/2204.01691.

[2] Jeannette Bohg, Marco Pavone, and Dorsa Sadigh. Markov decision processes, introduction to reinforcement learning. Lecture Notes, CS237B: Principles of Robot Autonomy II, Stanford University, 2024. URL https://web.stanford.edu/class/cs237b/pdfs/lecture/cs237b_lecture_3.pdf.

[3] Liangliang Chen, Yutian Lei, Shiyu Jin, Ying Zhang, and Liangjun Zhang. Rlingua: Improving reinforcement learning sample efficiency in robotic manipulations with large language models. *IEEE Robotics and Automation Letters*, 9(7):6075–6082, 2024.

[4] Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023. URL https://arxiv.org/abs/2305.14325.

[5] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models, 2022. URL https://arxiv.org/abs/2207.05608.

[6] Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution, 2025. URL https://arxiv.org/abs/2509.19349.

[7] Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate, 2024. URL https://arxiv.org/abs/2305.19118.

[8] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

[9] Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. Robocasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *International Conference on Learning Representations (ICLR)*, 2026.

[10] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

[11] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations, 2018. URL https://arxiv.org/abs/1709.10087.

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[13] Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. Deep reinforcement learning for robotics: A survey of real-world successes, 2024. URL https://arxiv.org/abs/2408.03539.

[14] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

[15] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, Yifeng Zhu, and Kevin Lin. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

APPENDIX

For comparison, we also include an early-stage policy generated at iteration 3, which reflects a simpler and less structured heuristic.

```
def policy(state, memory):
    n = state["num_blocks"]
    picked = state.get("picked", [])
    available = [i for i in range(n) if not picked[i
    ]]
    if available:
        return max(available)
    return 0
```

This policy illustrates early-stage bias toward high-index selection, prior to convergence toward structured swapping behavior.

ACKNOWLEDGMENTS

...

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, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can, not as i say: Grounding language in robotic affordances, 2022. URL https://arxiv.org/abs/2204.01691.

[2] Jeannette Bohg, Marco Pavone, and Dorsa Sadigh. Markov decision processes, introduction to reinforcement learning. Lecture Notes, CS237B: Principles of Robot Autonomy II, Stanford University, 2024. URL https://web.stanford.edu/class/cs237b/pdfs/lecture/cs237b_lecture_3.pdf.

[3] Liangliang Chen, Yutian Lei, Shiyu Jin, Ying Zhang, and Liangjun Zhang. Rlingua: Improving reinforcement learning sample efficiency in robotic manipulations with large language models. *IEEE Robotics and Automation Letters*, 9(7):6075–6082, 2024.

[4] Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate, 2023. URL https://arxiv.org/abs/2305.14325.

[5] Wenlong Huang, Fei Xia, Ted Xiao, Harris Chan, Jacky Liang, Pete Florence, Andy Zeng, Jonathan Tompson, Igor Mordatch, Yevgen Chebotar, Pierre Sermanet, Noah Brown, Tomas Jackson, Linda Luu, Sergey Levine, Karol

Hausman, and Brian Ichter. Inner monologue: Embodied reasoning through planning with language models, 2022. URL https://arxiv.org/abs/2207.05608.

[6] Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution, 2025. URL https://arxiv.org/abs/2509.19349.

[7] Tian Liang, Zhiwei He, Wenxiang Jiao, Xing Wang, Yan Wang, Rui Wang, Yujiu Yang, Shuming Shi, and Zhaopeng Tu. Encouraging divergent thinking in large language models through multi-agent debate, 2024. URL https://arxiv.org/abs/2305.19118.

[8] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

[9] Soroush Nasiriany, Sepehr Nasiriany, Abhiram Maddukuri, and Yuke Zhu. Robocasa365: A large-scale simulation framework for training and benchmarking generalist robots. In *International Conference on Learning Representations (ICLR)*, 2026.

[10] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

[11] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations, 2018. URL https://arxiv.org/abs/1709.10087.

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[13] Chen Tang, Ben Abbatematteo, Jiaheng Hu, Rohan Chandra, Roberto Martín-Martín, and Peter Stone. Deep reinforcement learning for robotics: A survey of real-world successes, 2024. URL https://arxiv.org/abs/2408.03539.

[14] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023. URL https://arxiv.org/abs/2201.11903.

[15] Yuke Zhu, Josiah Wong, Ajay Mandlekar, Roberto Martín-Martín, Abhishek Joshi, Soroush Nasiriany, Yifeng Zhu, and Kevin Lin. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.