

Discovering Interpretable Programmatic Policies via Multimodal LLM-assisted Evolutionary Search

Qinglong Hu¹, Xialiang Tong², Mingxuan Yuan², Fei Liu¹, Zhichao Lu¹, Qingfu Zhang¹

¹Department of Computer Science, City University of Hong Kong

²Huawei Noah's Ark Lab

qinglhu2-c@my.cityu.edu.hk, qingfu.zhang@cityu.edu.hk,

Abstract

Interpretability and high performance are essential goals in designing control policies, particularly for safety-critical tasks. Deep reinforcement learning has greatly enhanced performance, yet its inherent lack of interpretability often undermines trust and hinders real-world deployment. This work addresses these dual challenges by introducing a novel approach for programmatic policy discovery, called Multimodal Large Language Model-assisted Evolutionary Search (MLES). MLES utilizes multimodal large language models as policy generators, combining them with evolutionary mechanisms for automatic policy optimization. It integrates visual feedback-driven behavior analysis within the policy generation process to identify failure patterns and facilitate targeted improvements, enhancing the efficiency of policy discovery and producing adaptable, human-aligned policies. Experimental results show that MLES achieves policy discovery capabilities and efficiency comparable to Proximal Policy Optimization (PPO) across two control tasks, while offering transparent control logic and traceable design processes. This paradigm overcomes the limitations of predefined domain-specific languages, facilitates knowledge transfer and reuse, and is scalable across various control tasks. MLES shows promise as a leading approach for the next generation of interpretable control policy discovery.

1 Introduction

Interpretability and high performance are two central challenges in the design of policies for control tasks (Milani et al. 2024). In recent years, deep reinforcement learning (DRL) has achieved remarkable performance across various domains. However, two fundamental challenges persist, stemming from the neural network-based nature of DRL policies (Vouros 2022; Hickling et al. 2023; Puiutta and Veith 2020). First, such policies are opaque black boxes, offering limited transparency into their decision-making processes. This lack of interpretability undermines trust, verification, and adoption, particularly in safety-critical applications such as autonomous driving and healthcare (Perez-Cerrolaza et al. 2024). Second, the policy learning process in DRL relies on gradient-based optimization over high-dimensional parameter spaces, making it difficult to analyze, intervene in, or reuse the learned knowledge. These limitations have driven the ongoing pursuit of policies that are not only high-performing but also transparent, verifiable,

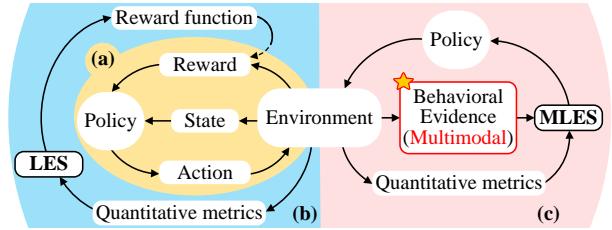


Figure 1: Overview of methodological differences. (a) Standard RL: agents learn via reward-guided interaction with environments. (b) Existing LES (LLM-assisted Evolutionary Search) in the RL domain: agents are improved indirectly by reshaping reward functions using LLMs. (c) Our MLES: directly evolves policies by integrating behavior analysis during the policy discovery process.

and human-understandable.

The recent rise of large language models (LLMs) offers a promising opportunity to develop policies that are both interpretable and high-performing. LLMs possess strong capabilities in contextual understanding, reasoning, and generation, particularly excelling in coding-related tasks (Achiam et al. 2023). Building on this foundation, the *LLM-assisted Evolutionary Search* (LES) paradigm has emerged as a powerful framework for automated design. LES combines the generative and reasoning capabilities of LLMs with the iterative optimization strengths of Evolutionary Computation (EC) (Liu et al. 2024a). In this paradigm, LLMs serve as mutation or crossover operators, guided by prompt templates and evaluation feedback, to iteratively refine candidate designs (Romera-Paredes et al. 2024; Liu et al. 2024a; Ye et al. 2024). LES has shown success in the automated discovery of code, algorithms, and heuristics. In the RL domain, recent studies such as *Eureka* et al. (Ma et al. 2024; Kwon et al. 2023; Sun et al. 2024; Masadome and Harada 2025) have demonstrated that LES can be used to evolve reward functions that improve agent performance.

Inspired by these advances, we pose the question of whether LES can directly synthesize functional and interpretable programmatic policies, rather than merely designing auxiliary components like reward functions. This direction holds the potential to yield agents that are both effective and transparent, thus bridging the gap between high-

performance black-box policies and human-understandable control logic. Building on this, we further investigate how LES can be enhanced to generate more reliable and generalizable policies, while also improving the efficiency of the policy discovery process.

To address these questions, we propose *Multimodal LLM-assisted Evolutionary Search* (MLES), a novel framework for the automatic discovery of programmatic policies. MLES extends the LES paradigm in two key ways. First, it targets the direct evolution of policies, where policies are expressed as executable programs enriched with natural language rationales. These are developed through natural language interactions with LLMs, enabling the synthesis of policies that are both semantically meaningful and easily interpretable. Second, MLES introduces *behavior analysis* into the evolutionary loop. By grounding policy evaluation in richer signals beyond scalar quantitative metrics, MLES enables more precise, adaptable, and human-aligned policy evolution. This enhances the search efficiency and makes the policy discovery process more transparent and traceable. In summary, this paper makes the following contributions:

(1) We propose the MLES framework, which integrates multimodal LLMs (MLLMs) with EC to directly synthesize interpretable programmatic policies through interaction with the environment. Unlike existing LES-based approaches that focus on reward shaping, MLES enables end-to-end policy discovery. An overview comparison is shown in Fig. 1.

(2) We present a prototypical instantiation of MLES and evaluate it on two standard RL benchmarks: Lunar Lander and Car Racing. Experimental results show that MLES produces effective policies with transparent control logic and traceable design processes, achieving performance comparable to Proximal Policy Optimization (PPO).

(3) We conduct extensive analyses of MLES in terms of effectiveness, search efficiency, and generalization, and investigate how different forms of behavioral evidence and prompt design influence the policy evolution process.

2 Related Work

2.1 Learning Interpretable Policies

Existing methods can be broadly categorized as either post hoc explanation and inherently interpretable modeling (Vouros 2022; Glanois et al. 2024). Post hoc methods attempt to explain trained black-box policies using human-understandable structures such as decision trees (Bastani, Pu, and Solar-Lezama 2018; Coppens et al. 2019; Gokhale et al. 2024; Kohler et al. 2024), mathematical formulas (Hein, Udluft, and Runkler 2018; Landajuela et al. 2021; Hazra and De Raedt 2023), finite-state machines (Koul, Greydanus, and Fern 2018; Inala et al. 2020), or domain-specific languages (DSLs) (Verma et al. 2018; Verma 2019). These methods typically rely on expert demonstrations or oracle supervision, using supervised or imitation learning to mimic the behavior (Verma et al. 2019; Cheng, Kolobov, and Agarwal 2020). Although achieving near-perfect performance, they offer limited insight into proactively constructing or improving policies, and tend to lack generalization beyond the oracle’s domain (Gu et al. 2025). Inherently

interpretable approaches, by contrast, aim to discover interpretable policies directly from environment interactions. These policies are represented as formulas (Hein, Udluft, and Runkler 2019; Lyu et al. 2019), logic rules (Jiang and Luo 2019; Delfosse et al. 2023; Glanois et al. 2022), decision trees (Silva et al. 2020; Topin et al. 2021; Pace, Chan, and van der Schaar 2022; Wu et al. 2024), and DSLs (Trivedi et al. 2021; Qiu and Zhu 2022; Liu et al. 2023; Gu et al. 2024; Liu et al. 2024b). Although these methods improve transparency, they are often constrained by static grammars, handcrafted templates, and limited symbolic search spaces, which restrict their expressiveness, adaptability, and scalability in complex environments compared to DRL. Many further assume program sketches (Cao et al. 2022), reducing flexibility across diverse control tasks.

Our proposed MLES framework belongs to the latter category but substantially broadens its scope in three key ways. First, it represents policies as executable programs augmented with natural language explanations, enabling both transparency and semantic introspection. Second, it learns purely from environment interaction, without reliance on expert data or oracle guidance. Third, it replaces static grammars or handcrafted operators with LLMs, allowing for the dynamic generation of expressive and diverse control logic through linguistic reasoning. These features collectively enable the synthesis of policies that are human-readable and capable of expressing more precise decision logic.

2.2 LLM-assisted Evolutionary Search

EC has long served as a powerful paradigm for black-box optimization, particularly in non-differentiable or combinatorial domains (Bäck, Fogel, and Michalewicz 1997). Recent advances have integrated LLMs into the evolutionary loop, giving rise to a new class of frameworks known as LES. By employing LLMs as evolutionary operators, LES shifts the search space from numerical or symbolic encodings to linguistic representations, enabling the evolution of code or other structured artifacts. This paradigm has demonstrated success in domains such as code generation (Hemberg, Moskal, and O’Reilly 2024), scientific discovery (Romera-Paredes et al. 2024; Shojaei et al. 2024), and algorithm design (Liu et al. 2024a; Ye et al. 2024; Aglietti et al. 2024). In RL, LES has been primarily applied to the indirect optimization of agents by evolving reward functions (Ma et al. 2024; Hazra et al. 2024), thereby improving the performance of downstream neural policies. However, the resulting policies remain neural and opaque.

In contrast, our work extends LES toward the direct synthesis of programmatic policies, shifting the focus from auxiliary reward function design to direct policy discovery from environment interaction. Furthermore, we incorporate behavior analysis into the evolutionary loop, allowing the search to be guided by richer information beyond scalar rewards. This enables a more informed and transparent form of policy discovery, in which both the resulting policy and its design rationale are accessible for inspection. To the best of our knowledge, this is the first known effort to unify LLMs, evolutionary search, and multimodal evaluation for the direct synthesis of interpretable policies in RL tasks.

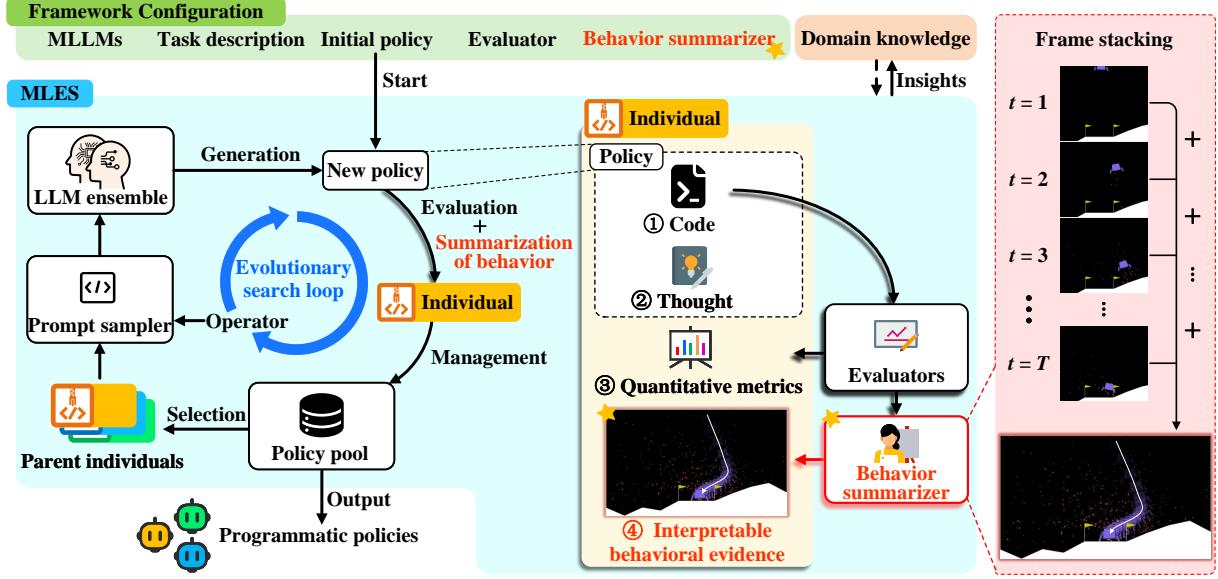


Figure 2: An overview of the MLES framework. The left side of the MLES illustrates the evolutionary search loop, while the right side details the structure and construction of an evolutionary individual. The red module on the far right exemplifies a method for generating behavioral evidence. During each search step, a subset of parent individuals is selected from a policy pool and used by the prompt sampler to create a multimodal few-shot prompt. MLLMs reason over this prompt to generate a new offspring policy. The offspring is then evaluated and visualized, resulting in the creation of a new individual that is added to the policy pool and managed accordingly. This iterative process continues, enabling the ongoing refinement of the policies.

2.3 Multimodal Large Language Models

MLLMs extend the capabilities of LLMs by integrating visual and textual modalities, enabling grounded understanding and vision-language reasoning (Caffagni et al. 2024). Recent models such as GPT-4V (Hurst et al. 2024) and Qwen2.5-VL (Bai et al. 2025) demonstrate strong performance on tasks requiring joint visual-language comprehension for planning, decision-making, and high-level control. These advances have led to increasing applications of MLLMs in planning and optimization. Some approaches employ MLLMs for real-time action selection and short-horizon planning in interactive environments (Zheng et al. 2024; Szot et al. 2025). Others leverage MLLMs as optimizers that iteratively refine solutions through visual feedback (Elhenawy et al. 2024a,b; Zhao and Cheong 2025), or use them for scene understanding in simulated and real-world domains (Li et al. 2025; Kambara et al. 2024; Hori et al. 2025). More recently, MLLMs have also been used to adapt reward functions based on visual cues (Narin 2024; Wang et al. 2025; Cuzin-Rambaud et al. 2025), thereby improving agent learning in complex RL settings.

These developments suggest that MLLMs are now capable of combining visual perception with contextual reasoning, a key capacity that aligns closely with our demands of interpretable policy discovery. In the MLES framework, we leverage the vision-language capabilities of MLLMs to guide the policy search process with rich information, including policy execution traces, in-depth analyses, and promising corrective suggestions. This approach facilitates a more transparent and traceable policy discovery, ultimately

supporting the synthesis of reliable, human-aligned policies.

3 Methodology

This section presents the MLES framework, designed to automatically discover high-performing and interpretable programmatic policies for control tasks.

3.1 Problem Definition

Policy discovery refers to the process of searching for high-quality policies within a predefined policy space, with the goal of maximizing expected performance in a given environment. Formally, given a policy space Π and an evaluation function $F(\pi)$ that measures the quality of a policy $\pi \in \Pi$, the policy discovery problem can be formulated as the following optimization problem:

$$\pi^* = \arg \max_{\pi \in \Pi} F(\pi) \quad (1)$$

Different policy discovery methods achieve this by varying the policy space Π and the optimization mechanisms used. For instance, DRL methods employ optimizers to search for optimal neural network parameters within a continuous parameter space. Genetic Programming combined with DSLs uses evolutionary algorithms to explore and identify the optimal combinations of predefined grammars within a discrete grammar space.

In the case of MLES, policy discovery is framed as a search process driven by LLMs within a knowledge-rich policy space. When given a control task, LLMs implicitly and inherently shape this policy space based on their understanding of the task. This space is rich with task-relevant

concepts and ideas, as well as valuable insights from other fields, allowing for flexible and expressive policy generation. By integrating MLLMs as policy generators within an evolutionary search process, MLES allows for purposeful exploration and exploitation of the policy space, facilitating the progressive discovery of higher-performing policies.

3.2 Framework Overview

The MLES framework orchestrates a closed-loop evolutionary process to iteratively search for programmatic policies within the policy space, as illustrated in Fig. 2. The framework requires the following configurable inputs to facilitate the discovery: (1) a selection of MLLMs for policy generation, (2) a formal task description, (3) an initial set of programmatic policies, (4) an evaluator for executing policies in the environment, and (5) a summarizer for generating interpretable behavioral evidence from policy executions. The framework consists of the following six core components:

Evolutionary search loop. The evolutionary search loop serves as the main control mechanism that orchestrates the entire evolutionary process. It manages iterations and maintains a dynamic balance between exploration and exploitation via configurable evolutionary operators, ensuring an effective search within the policy space.

Policy pool. The policy pool acts as a dynamic repository that holds the current population of policies. It is responsible for selecting appropriate parent candidates based on the evolutionary operator and integrating newly generated offspring into the population, thereby facilitating the continuous evolution of the policy set.

Evaluators. The evaluators are responsible for executing policies in the target environment. They output both quantitative performance metrics (e.g., episode rewards) and raw behavior traces. To improve efficiency and minimize delays in the policy search process, this evaluation procedure can be conducted in parallel.

Summarizer. The summarizer converts raw behavior traces collected by the evaluators into interpretable behavioral evidence. This evidence captures qualitative aspects of policy execution that go beyond scalar performance metrics, offering a deeper understanding of the policy’s behavior. This component is crucial for enabling informed policy modifications by the MLLMs and represents a key innovation of MLES.

Prompt sampler. The prompt sampler constructs multimodal few-shot prompts tailored to the current evolutionary operator by integrating selected parent policies and their corresponding interpretable behavioral evidence. These prompts provide essential context and guidance, directing the MLLMs to propose meaningful policies aligned with the intended transformation, thus steering the evolution toward promising directions.

LLMs ensemble. This component consists of a configurable set of MLLMs. These MLLMs reason over the few-shot prompts, leveraging their advanced vision-language un-

derstanding, reasoning, and code generation capabilities to generate new candidate policies.

Through the iterative refinement process, MLES progressively discovers a diverse set of interpretable and high-performing programmatic policies. Concurrently, this process explores a wide range of both successful and suboptimal policies, yielding valuable insights that enhance human understanding of the target task and provide a foundation for knowledge transfer and reuse in related control scenarios.

3.3 Policy Representation

During the search process, each candidate policy is treated as an individual, structured as a tuple consisting of four elements that capture the policy’s logic, intent, performance, and behavioral characteristics: Code, Thought, Quantitative Metrics, and Interpretable Behavioral Evidence (IBE).

Code. Each policy is implemented as an executable Python program that defines the agent’s decision-making logic. This representation supports automated evaluation within the evolutionary loop and ensures direct compatibility with the environment. Moreover, expressing policies in code offers high transparency and modularity, making them human-readable, easy to debug, and conducive to reuse or modification during the search process.

Thought. The thought is a concise natural language summary that captures the underlying rationale or design intent of a policy, highlighting key strategies that guide the agent’s behavior. By providing a high-level semantic abstraction, thoughts help LLMs interpret the associated code more effectively and support deeper reasoning during policy generation. Prior work (Liu et al. 2024a) has shown that incorporating such descriptions into few-shot prompts can significantly improve the efficiency and effectiveness of LES.

Quantitative metrics. Quantitative metrics are task-specific numerical evaluations obtained by executing the policy in the environment. These metrics serve as direct indicators of policy’s performance (i.e., $F(\cdot)$ in 3.1), such as episode rewards, success rates, or other user-defined scores tailored to the tasks. They provide a consistent basis for ranking and selecting candidate policies, ensuring high-performing individuals are effectively identified and prioritized for further refinement.

IBE. While quantitative metrics are essential for measuring performance, they provide only a partial understanding of a policy’s behavior pattern and often fail to offer actionable guidance for improvement. To address this limitation, MLES introduces IBE, which captures the overall behavior of a policy or highlights key events that help identify areas for improvement. By analyzing IBE, MLLMs can observe policy outcomes in detail, accurately identifying failure modes such as incorrect decisions or reward hacking (e.g., exploiting bugs for undeserved rewards), thus enabling more targeted policy modifications and corrections. This provides richer information for MLLMs to reason over, facilitating the discovery of human-aligned policies. This design mirrors how human experts refine policies: they do not merely rely on evaluating numerical scores but also examine

behavioral patterns (e.g., failure modes, unexpected interactions) to identify areas for improvement. The form of IBE is flexible, including images, videos, and even text-based state sequences, and more. Examples of these formats and more detailed descriptions can be found in Appendix C.

3.4 Evolutionary Search for Discovering Policy

This subsection details the evolutionary search loop in MLES. Building upon the EoH framework (Liu et al. 2024a), we integrate behavioral evidence analysis, thereby introducing a prototype implementation of MLES designed to discover high-performing programmatic policies. At each search step, parent selection determines the starting points, while evolutionary operators guide the sampling of new policies. By combining carefully designed operators with principled selection strategies, MLES achieves a dynamic balance between exploration and exploitation, facilitating the search for high-performing policies within the policy space.

Evolutionary operators. The core of MLES is a set of evolutionary operators that drive the generation of new policies. We define four operators, categorized into two types:

Exploration operators (E1 and E2): These operators aim to enhance the behavioral diversity of the population by generating novel policies based on the analysis of parent policies’ control logic. **E1** prompts the LLM to examine both the code and thought of selected parent policies and then synthesize a new policy that implements fundamentally different control strategies, encouraging exploration of uncharted regions in the policy space. **E2** prompts the LLM to identify shared patterns across multiple parent policies and construct a new policy that generalizes from these patterns, akin to the crossover operation in evolutionary computation.

Multimodal modification operators (M1_M and M2_M): These operators instruct MLLMs to analyze IBE and refine existing policies through detailed, informed modification. **M1_M** asks the MLLMs to identify behavioral shortcomings by jointly analyzing the policy code and its IBE, then revise the control logic accordingly. **M2_M** prompts the MLLMs to identify critical parameters in the policy and adjust them based on the observed evidence, leading to targeted optimization. The integration of IBE analysis provides grounded insights for the MLLM’s reasoning, resulting in more coherent and purposeful offspring generation.

Parent selection. For each operator, a subset of candidate policies is selected from the policy pool to serve as parents. The selection follows a rank-based probabilistic scheme common in evolutionary computation. Specifically, the selection probability of the i -th policy is defined as $p_i \propto 1/(r_i + N)$, where r_i denotes the rank of the policy (according to quantitative metrics), and N is the size of the policy pool. This scheme encourages the exploration of elite policies while maintaining diversity by occasionally selecting lower-ranked individuals.

Few-shot prompt construction. After selecting the operator and parents, the prompt sampler constructs a task-specific few-shot prompt to guide the MLLMs in generating

new policies. Each prompt consists of the following components: (1) a task description, (2) the code and thought of each parent policy, (3) relevant IBE, and (4) operator-specific instructions. Together, these elements provide the necessary context and guidance for effective policy generation.

Policy generation & individual construction. Guided by the constructed prompts, the LLM ensemble generates multiple candidate policies in parallel. These policies are then executed by evaluators to assess their quantitative metrics. The raw data from these executions is processed by the behavior summarizer, converting it into IBE, and ultimately leading to the construction of the corresponding individual.

Policy pool management. Offspring individuals are filtered for redundancy and ranked based on their quantitative metrics. The top-performing and diverse candidates are then added back into the policy pool, enabling progressive refinement and sustained diversity over successive iterations.

4 Experiments

We conduct empirical experiments to evaluate the effectiveness, efficiency, interpretability, and generalization capabilities of MLES.

4.1 Experimental Setting

Benchmarks. We evaluate the proposed MLES framework on two representative control tasks from the OpenAI Gym suite (Brockman et al. 2016): **Lunar Lander** and **Car Racing**. These tasks collectively cover both discrete and continuous control settings, providing a comprehensive testbed for our method.

- **Lunar Lander** is a discrete-action control task where the agent must land a spacecraft on a designated pad. The agent has four possible discrete actions: doing nothing, applying thrust from the main engine, or firing the left/right orientation engines. The environment has an 8-dimensional state space and is commonly used for benchmarking RL methods in discrete domains.
- **Car Racing** is a more complex continuous-control task. The agent drives a car along procedurally generated tracks based on pixel observations. This control task poses a greater challenge due to its high-dimensional input ($96 \times 96 \times 3$ image) and continuous action space, making it an ideal testbed for evaluating image-conditioned programmatic policies.

Baselines. We compare MLES against both DRL methods and an existing LES approach for direct policy discovery:

- **Deep Q-Network (DQN)** (Mnih et al. 2013) is a value-based DRL algorithm designed for discrete action spaces. It serves as a representative DRL baseline for Lunar Lander.
- **Proximal Policy Optimization (PPO)** (Schulman et al. 2017) is a widely used on-policy policy gradient method that can handle both discrete and continuous control tasks.

- EoH (Liu et al. 2024a) is a recent LES framework for automated code discovery. We use it as a baseline to assess the added value of behavioral evidence analysis in MLES, as it shares the same evolutionary framework but lacks the visual feedback-driven behavior analysis used in MLES.

Implementation Details. For the Lunar Lander task, we select five representative instances for training, while for the Car Racing task, four representative tracks are chosen. Both tasks use ten test environments, each generated from a random seed in the range zero to nine. The quantitative metric used for evaluating performance in the Lunar Lander task is the normalized weight score (NWS), defined as:

$$NWS = \frac{R}{200} \times 0.6 + (1 - \min(\frac{C}{100}, 1)) \times 0.2 + S \times 0.2 \quad (2)$$

where R is the mean episode reward, C is the mean fuel consumption, and S is the success rate of completing the landing task, all computed across multiple instances. For the Car Racing task, we use the average track completion rate across instances as the quantitative metric. For DRL methods, we adopt the default reward function provided by Gym. Further details on these metrics are available in Appendix B.

Both MLES and EoH employ GPT-4o-mini for policy generation. The LLM query budget is set to 2000 requests, corresponding to 10,000 and 8,000 environment resets for Lunar Lander and Car Racing, respectively. To ensure a fair comparison, we train all baselines under an identical budget of environment resets. The evolutionary search population size is set to 16, with the initial population of policies for MLES shared with EoH to avoid any biases arising from different starting points. All experiments are conducted on a machine equipped with an Intel Core i9-13980HX processor and 32 GB of RAM. Each experiment is repeated five times to ensure the robustness and reliability of the results.

4.2 Experimental Results

Method	Lunar Lander		Car Racing	
	Train	Test	Train	Test
DQN	1.017	0.508	79.777	71.724
PPO	1.032	0.846	99.212	<u>94.546</u>
Initial Policy	0.629	0.653	17.772	17.619
EoH	<u>1.053</u>	0.776	89.808	79.286
MLES	1.090	<u>0.819</u>	<u>98.704</u>	96.358

Table 1: Performance on benchmarks (5 runs). Best results are in **bold**, second-best are underlined. Higher is better.

Comparative evaluations of MLES Table 1 reports the average performance across five independent runs for each method on both training and testing instances. MLES gets the highest training performance on Lunar Lander and ranks as the second-best on Car Racing, closely trailing the PPO baseline. Compared to the Initial Policy, which serves as the

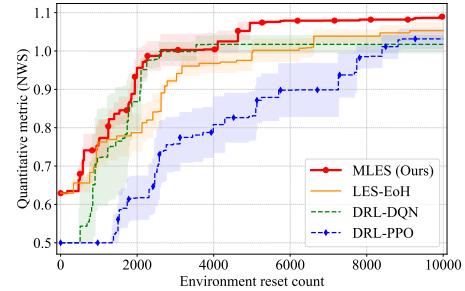


Figure 3: Performance convergence on Lunar Lander task

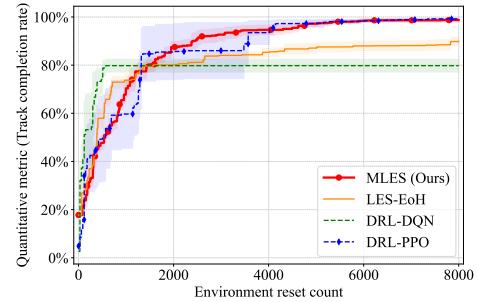


Figure 4: Performance convergence on Car Racing Task

search starting point, MLES achieves substantial improvements across both tasks, demonstrating its practical effectiveness. These results provide a positive answer to our first research question: MLES is capable of directly synthesizing high-performing policies through environment interaction, delivering performance on par with strong DRL baselines.

Compared to EoH, MLES consistently outperforms it on both training and testing instances. The performance gap is especially notable on the more challenging Car Racing task, indicating that integrating behavior analysis significantly enhances the effectiveness and robustness of evolved policies.

However, we observe a consistent drop in test performance relative to training across all methods. This indicates that generalization remains a challenge, especially in high-variance environments. In Section 4.3, we explore how MLES can leverage its population-based nature to improve generalization through policy ensembling.

Analysis of policy discovery efficiency Figs. 3 and 4 illustrate the convergence of the best policy performance with respect to the count of environment resets for four methods. Colored curves represent the mean performance over five runs, and shaded areas indicate the standard error of the mean, reflecting the stability of each method during the policy discovery process. MLES consistently demonstrates superior discovery efficiency compared to both DRL methods (DQN, PPO) and the LES-based baseline (EoH). On the simpler Lunar Lander task, MLES discovers high-performing policies within approximately 5,000 environment resets, which is substantially faster than both PPO and DQN. On the more challenging Car Racing task, MLES achieves a convergence speed comparable to PPO while exhibiting significantly lower variance across runs, indicat-

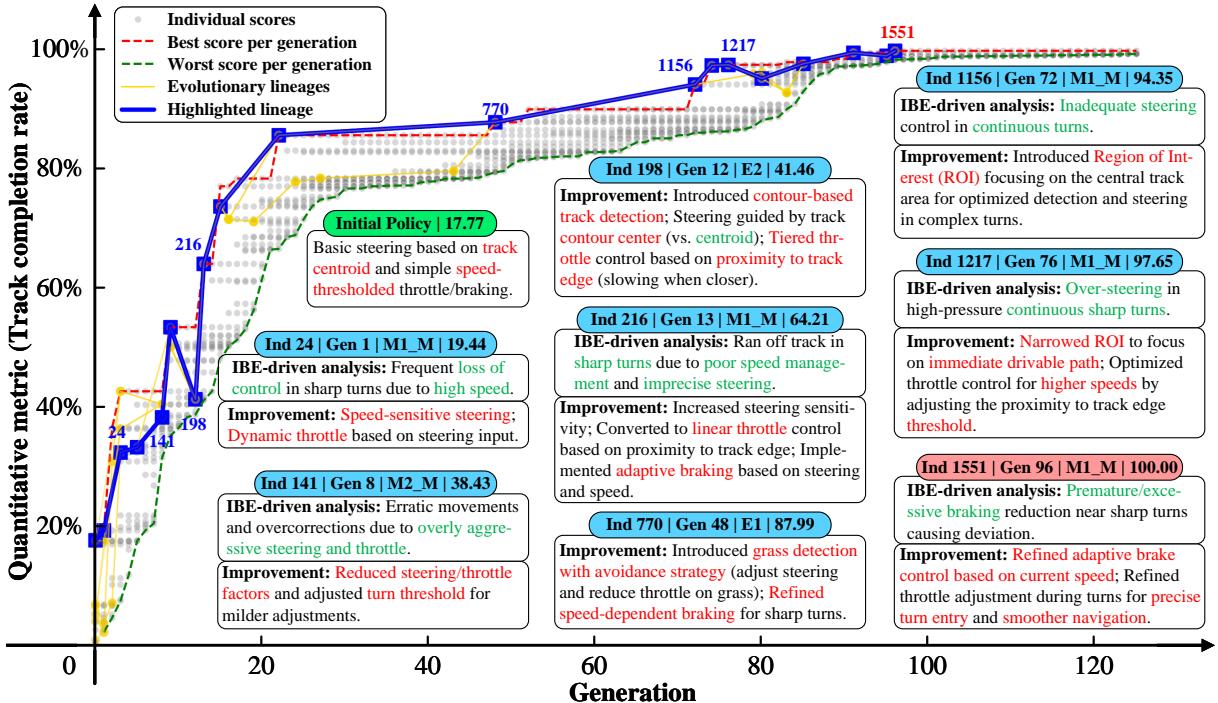


Figure 5: Evolutionary dynamics of Car Racing policies. The plot depicts population score distributions over generations, with yellow lines tracing all ancestors of the best-performing policy. The blue-highlighted lineage is examined in detail to reveal the stepwise improvements guided by IBE-driven insights.

ing more stable learning dynamics. In comparison to EoH, MLES yields both faster convergence and higher final performance across tasks. The narrower confidence intervals in later stages further emphasize MLES’s robustness and consistency. These results highlight the benefits of incorporating visual feedback-driven behavior analysis into the evolutionary process. By leveraging richer signals beyond scalar performance metrics, MLES more effectively guides policy search, leading to faster convergence and enhanced stability.

Another noteworthy point is that both MLES and EoH exhibit higher initial performance compared to DRL baselines. This advantage arises from the ability of the LES paradigm to easily use prior knowledge as initial policies, allowing for a more informed starting point. In contrast, defining meaningful policies in the form of neural networks is challenging, and DRL methods typically rely on randomly initialized weights. This highlights the inherent capacity of MLES to reuse and transfer knowledge. This capability is particularly valuable in domains where expert heuristics are available, enabling more sample-efficient exploration from the outset.

Qualitative analysis of the interpretability Figure 5 presents an example of the evolutionary process for Car Racing policies, showing how individual scores change throughout the discovery. This visualization clarifies why each policy was generated and how it was refined. Such transparency firmly demonstrates that the policy discovery process of MLES is interpretable and traceable. By analyzing the IBE, MLLMs identify failure patterns in parent policies and perform targeted improvements accordingly. The final policy’s

code and thought are provided in Appendix F. In summary, MLES offers interpretability on two levels: (1) the discovered programmatic policies are fully human-readable; (2) the policy discovery process is entirely transparent and highly informative for further research.

4.3 Generalization via Policy Ensemble

Method	Lunar Lander		Car Racing	
	Train	Test	Train	Test
EoH	1.053	0.776	89.808	79.286
EoH [†]	0.938↓	0.856↑	82.978↓	84.567↑
MLES	1.090	0.819	98.704	96.358
MLES [†]	1.032↓	0.901↑	96.904↓	97.921↑

Table 2: Effect of Policy Ensemble on LES Performance

As illustrated in Fig. 5, MLES generates a diverse population of high-performing policies in a single run. This opens up new avenues for improving generalization, particularly by applying ensemble decision-making techniques to these high-performing policies. Table 2 presents the results of a simple ensemble strategy, where discrete actions are voted on and continuous actions are averaged. Here, MLES[†] and EoH[†] denote the versions that employ ensemble decisions based on the entire population of policies.

Despite its simplicity, this ensemble approach significantly enhances test performance for both MLES and EoH,

highlighting the potential of ensemble decision-making to boost generalization. Notably, a slight decline in training performance is observed after ensembling. This outcome aligns with the well-known bias-variance tradeoff in machine learning, where ensembling generally improves generalization at the cost of slightly increased bias. In future work, we plan to explore more advanced techniques for enhancing generalization, such as incorporating instance feature awareness and adaptive collaborative decision-making. Overall, MLES shows inherent advantages and substantial potential in addressing the generalization issues.

A statistical analysis of Table 1 and an investigation into how different forms of IBE and prompts influence the policy discovery process are presented in Appendix D.

5 Conclusion

This paper presents Multimodal Large Language Model-assisted Evolutionary Search (MLES), which combines multimodal large language models with evolutionary computation to efficiently design interpretable programmatic policy for control tasks. By introducing visual feedback-driven behavior analysis, MLES mimics the process of human policy discovery. The framework is demonstrated on two standard control tasks, showcasing its policy discovery capability and efficiency comparable to Proximal Policy Optimization (PPO). Notably, MLES yields transparent control logic with traceable policy design processes. MLES offers a flexible and automated approach that reduces human effort and facilitates knowledge reuse, presenting a new paradigm for the next generation of interpretable control policy discovery.

References

- Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F. L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.
- Aglietti, V.; Ktena, I.; Schrouff, J.; Sgouritsa, E.; Ruiz, F. J.; Malek, A.; Bellot, A.; and Chiappa, S. 2024. FunBO: Discovering Acquisition Functions for Bayesian Optimization with FunSearch. *arXiv preprint arXiv:2406.04824*.
- Bäck, T.; Fogel, D. B.; and Michalewicz, Z. 1997. Handbook of evolutionary computation. *Release*, 97(1): B1.
- Bai, S.; Chen, K.; Liu, X.; Wang, J.; Ge, W.; Song, S.; Dang, K.; Wang, P.; Wang, S.; Tang, J.; et al. 2025. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*.
- Bastani, O.; Pu, Y.; and Solar-Lezama, A. 2018. Verifiable reinforcement learning via policy extraction. *Advances in neural information processing systems*, 31.
- Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; and Zaremba, W. 2016. Openai gym. *arXiv preprint arXiv:1606.01540*.
- Caffagni, D.; Cocchi, F.; Barsellotti, L.; Moratelli, N.; Sarto, S.; Baraldi, L.; Cornia, M.; and Cucchiara, R. 2024. The revolution of multimodal large language models: a survey. *arXiv preprint arXiv:2402.12451*.
- Cao, Y.; Li, Z.; Yang, T.; Zhang, H.; Zheng, Y.; Li, Y.; Hao, J.; and Liu, Y. 2022. GALOIS: boosting deep reinforcement learning via generalizable logic synthesis. *Advances in Neural Information Processing Systems*, 35: 19930–19943.
- Cheng, C.-A.; Kolobov, A.; and Agarwal, A. 2020. Policy improvement via imitation of multiple oracles. *Advances in Neural Information Processing Systems*, 33: 5587–5598.
- Coppens, Y.; Efthymiadis, K.; Lenaerts, T.; Nowé, A.; Miller, T.; Weber, R.; and Magazzeni, D. 2019. Distilling deep reinforcement learning policies in soft decision trees. In *Proceedings of the IJCAI 2019 workshop on explainable artificial intelligence*, 1–6.
- Cuzin-Rambaud, V.; Komlenovic, E.; Faure, A.; and Yun, B. 2025. VIRAL: Vision-grounded Integration for Reward design And Learning. *arXiv preprint arXiv:2505.22092*.
- Delfosse, Q.; Shindo, H.; Dhami, D.; and Kersting, K. 2023. Interpretable and explainable logical policies via neurally guided symbolic abstraction. *Advances in Neural Information Processing Systems*, 36: 50838–50858.
- Elhenawy, M.; Abdelhay, A.; Alhadidi, T. I.; Ashqar, H. I.; Jaradat, S.; Jaber, A.; Glaser, S.; and Rakotonirainy, A. 2024a. Eyeballing combinatorial problems: A case study of using multimodal large language models to solve traveling salesman problems. In *International Conference on Intelligent Systems, Blockchain, and Communication Technologies*, 341–355. Springer.
- Elhenawy, M.; Abutahoun, A.; Alhadidi, T. I.; Jaber, A.; Ashqar, H. I.; Jaradat, S.; Abdelhay, A.; Glaser, S.; and Rakotonirainy, A. 2024b. Visual Reasoning and Multi-Agent Approach in Multimodal Large Language Models (MLLMs): Solving TSP and mTSP Combinatorial Challenges. *arXiv preprint arXiv:2407.00092*.
- Glanois, C.; Jiang, Z.; Feng, X.; Weng, P.; Zimmer, M.; Li, D.; Liu, W.; and Hao, J. 2022. Neuro-symbolic hierarchical rule induction. In *International Conference on Machine Learning*, 7583–7615. PMLR.
- Glanois, C.; Weng, P.; Zimmer, M.; Li, D.; Yang, T.; Hao, J.; and Liu, W. 2024. A survey on interpretable reinforcement learning. *Machine Learning*, 113(8): 5847–5890.
- Gokhale, G.; Karimi Madahi, S. S.; Claessens, B.; and Develder, C. 2024. Distill2Explain: Differentiable decision trees for explainable reinforcement learning in energy application controllers. In *Proceedings of the 15th ACM International Conference on Future and Sustainable Energy Systems*, 55–64.
- Gu, Y.; Zhang, K.; Liu, Q.; Gao, W.; Li, L.; and Zhou, J. 2024. π -light: Programmatic interpretable reinforcement learning for resource-limited traffic signal control. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 21107–21115.
- Gu, Y.; Zhang, K.; Liu, Q.; Yu, R.; Lin, X.; and Sun, X. 2025. ProCC: Programmatic Reinforcement Learning for Efficient and Transparent TCP Congestion Control. In *Proceedings of the Eighteenth ACM International Conference on Web Search and Data Mining*, 963–972.
- Hazra, R.; and De Raedt, L. 2023. Deep explainable relational reinforcement learning: a neuro-symbolic approach. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 213–229. Springer.

- Hazra, R.; Sykounas, A.; Persson, A.; Loutfi, A.; and Martires, P. Z. D. 2024. REvolve: Reward Evolution with Large Language Models using Human Feedback. *arXiv preprint arXiv:2406.01309*.
- Hein, D.; Udluft, S.; and Runkler, T. A. 2018. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence*, 76: 158–169.
- Hein, D.; Udluft, S.; and Runkler, T. A. 2019. Generating interpretable reinforcement learning policies using genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, 23–24.
- Hemberg, E.; Moskal, S.; and O'Reilly, U.-M. 2024. Evolving code with a large language model. *Genetic Programming and Evolvable Machines*, 25(2): 21.
- Hickling, T.; Zenati, A.; Aouf, N.; and Spencer, P. 2023. Explainability in deep reinforcement learning: A review into current methods and applications. *ACM Computing Surveys*, 56(5): 1–35.
- Hori, C.; Kambara, M.; Sugiura, K.; Ota, K.; Khurana, S.; Jain, S.; Corcodel, R.; Jha, D.; Romeres, D.; and Le Roux, J. 2025. Interactive robot action replanning using multimodal llm trained from human demonstration videos. In *ICASSP 2025-2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 1–5. IEEE.
- Hurst, A.; Lerer, A.; Goucher, A. P.; Perelman, A.; Ramesh, A.; Clark, A.; Ostrow, A.; Welihinda, A.; Hayes, A.; Radford, A.; et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.
- Inala, J. P.; Bastani, O.; Tavares, Z.; and Solar-Lezama, A. 2020. Synthesizing programmatic policies that inductively generalize. In *8th International Conference on Learning Representations*.
- Jiang, Z.; and Luo, S. 2019. Neural logic reinforcement learning. In *International conference on machine learning*, 3110–3119. PMLR.
- Kambara, M.; Hori, C.; Sugiura, K.; Ota, K.; Jha, D. K.; Khurana, S.; Jain, S.; Corcodel, R.; Romeres, D.; and Le Roux, J. 2024. Human action understanding-based robot planning using multimodal llm. In *IEEE International Conference on Robotics and Automation (ICRA) Workshop*.
- Kohler, H.; Delfosse, Q.; Akrour, R.; Kersting, K.; and Preux, P. 2024. Interpretable and editable programmatic tree policies for reinforcement learning. *arXiv preprint arXiv:2405.14956*.
- Koul, A.; Greydanus, S.; and Fern, A. 2018. Learning finite state representations of recurrent policy networks. *arXiv preprint arXiv:1811.12530*.
- Kwon, M.; Xie, S. M.; Bullard, K.; and Sadigh, D. 2023. Reward design with language models. *arXiv preprint arXiv:2303.00001*.
- Landajuela, M.; Petersen, B. K.; Kim, S.; Santiago, C. P.; Glatt, R.; Mundhenk, N.; Pettit, J. F.; and Faissol, D. 2021. Discovering symbolic policies with deep reinforcement learning. In *International Conference on Machine Learning*, 5979–5989. PMLR.
- Li, Z.; Xi-Jia, Z.; Altundas, B.; Chen, L.; Paleja, R.; and Gombolay, M. 2025. Towards Automated Semantic Interpretability in Reinforcement Learning via Vision-Language Models. *arXiv preprint arXiv:2503.16724*.
- Liu, F.; Xialiang, T.; Yuan, M.; Lin, X.; Luo, F.; Wang, Z.; Lu, Z.; and Zhang, Q. 2024a. Evolution of Heuristics: Towards Efficient Automatic Algorithm Design Using Large Language Model. In *Forty-first International Conference on Machine Learning*.
- Liu, G.-T.; Hu, E.-P.; Cheng, P.-J.; Lee, H.-Y.; and Sun, S.-H. 2023. Hierarchical programmatic reinforcement learning via learning to compose programs. In *International Conference on Machine Learning*, 21672–21697. PMLR.
- Liu, M.; Yu, C.-H.; Lee, W.-H.; Hung, C.-W.; Chen, Y.-C.; and Sun, S.-H. 2024b. Synthesizing programmatic reinforcement learning policies with large language model guided search. *arXiv preprint arXiv:2405.16450*.
- Lyu, D.; Yang, F.; Liu, B.; and Gustafson, S. 2019. SDRL: interpretable and data-efficient deep reinforcement learning leveraging symbolic planning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 2970–2977.
- Ma, Y. J.; Liang, W.; Wang, G.; Huang, D.-A.; Bastani, O.; Jayaraman, D.; Zhu, Y.; Fan, L.; and Anandkumar, A. 2024. Eureka: Human-Level Reward Design via Coding Large Language Models. In *ICLR*.
- Masadome, S.; and Harada, T. 2025. Reward design using large language models for natural language explanation of reinforcement learning agent actions. *IEEJ Transactions on Electrical and Electronic Engineering*.
- Milani, S.; Topin, N.; Veloso, M.; and Fang, F. 2024. Explainable reinforcement learning: A survey and comparative review. *ACM Computing Surveys*, 56(7): 1–36.
- Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.
- Narin, A. 2024. Evolutionary reward design and optimization with multimodal large language models. In *Proceedings of the 3rd Workshop on Advances in Language and Vision Research (ALVR)*, 202–208.
- Pace, A.; Chan, A. J.; and van der Schaar, M. 2022. Poetree: Interpretable policy learning with adaptive decision trees. *arXiv preprint arXiv:2203.08057*.
- Perez-Cerrolaza, J.; Abella, J.; Borg, M.; Donzella, C.; Cerquides, J.; Cazorla, F. J.; Englund, C.; Tauber, M.; Nikolakopoulos, G.; and Flores, J. L. 2024. Artificial intelligence for safety-critical systems in industrial and transportation domains: A survey. *ACM Computing Surveys*, 56(7): 1–40.
- Puiutta, E.; and Veith, E. M. 2020. Explainable reinforcement learning: A survey. In *International cross-domain conference for machine learning and knowledge extraction*, 77–95. Springer.
- Qiu, W.; and Zhu, H. 2022. Programmatic reinforcement learning without oracles. In *The Tenth International Conference on Learning Representations*.

- Romera-Paredes, B.; Barekatain, M.; Novikov, A.; Balog, M.; Kumar, M. P.; Dupont, E.; Ruiz, F. J.; Ellenberg, J. S.; Wang, P.; Fawzi, O.; et al. 2024. Mathematical discoveries from program search with large language models. *Nature*, 625(7995): 468–475.
- Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.
- Shojaee, P.; Meidani, K.; Gupta, S.; Farimani, A. B.; and Reddy, C. K. 2024. LLM-SR: Scientific Equation Discovery via Programming with Large Language Models. *arXiv preprint arXiv:2404.18400*.
- Silva, A.; Gombolay, M.; Killian, T.; Jimenez, I.; and Son, S.-H. 2020. Optimization methods for interpretable differentiable decision trees applied to reinforcement learning. In *International conference on artificial intelligence and statistics*, 1855–1865. PMLR.
- Skalse, J.; Howe, N.; Krasheninnikov, D.; and Krueger, D. 2022. Defining and characterizing reward gaming. *Advances in Neural Information Processing Systems*, 35: 9460–9471.
- Sun, S.; Liu, R.; Lyu, J.; Yang, J.-W.; Zhang, L.; and Li, X. 2024. A Large Language Model-Driven Reward Design Framework via Dynamic Feedback for Reinforcement Learning. *arXiv preprint arXiv:2410.14660*.
- Szot, A.; Mazoure, B.; Attia, O.; Timofeev, A.; Agrawal, H.; Hjelm, D.; Gan, Z.; Kira, Z.; and Toshev, A. 2025. From multimodal ILMs to generalist embodied agents: Methods and lessons. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, 10644–10655.
- Topin, N.; Milani, S.; Fang, F.; and Veloso, M. 2021. Iterative bounding mdps: Learning interpretable policies via non-interpretable methods. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 9923–9931.
- Trivedi, D.; Zhang, J.; Sun, S.-H.; and Lim, J. J. 2021. Learning to synthesize programs as interpretable and generalizable policies. *Advances in neural information processing systems*, 34: 25146–25163.
- Verma, A. 2019. Verifiable and interpretable reinforcement learning through program synthesis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 9902–9903.
- Verma, A.; Le, H.; Yue, Y.; and Chaudhuri, S. 2019. Imitation-projected programmatic reinforcement learning. *Advances in Neural Information Processing Systems*, 32.
- Verma, A.; Murali, V.; Singh, R.; Kohli, P.; and Chaudhuri, S. 2018. Programmatically interpretable reinforcement learning. In *International conference on machine learning*, 5045–5054. PMLR.
- Vouros, G. A. 2022. Explainable deep reinforcement learning: state of the art and challenges. *ACM Computing Surveys*, 55(5): 1–39.
- Wang, K.; Zhao, Y.; He, Y.; Dai, S.; Zhang, N.; and Yang, M. 2025. Guiding reinforcement learning with shaping rewards provided by the vision–language model. *Engineering Applications of Artificial Intelligence*, 155: 111004.
- Wang, Y.; and Venkatesh, G. 2025. Read Quietly, Think Aloud: Decoupling Comprehension and Reasoning in LLMs. *arXiv preprint arXiv:2507.03327*.
- Wu, T.; Shen, L.; Dong, Z.; Peng, X.; and Zhao, W. 2024. Synthesizing programmatic policy for generalization within task domain. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, 5217–5225.
- Ye, H.; Wang, J.; Cao, Z.; Berto, F.; Hua, C.; Kim, H.; Park, J.; and Song, G. 2024. Reevo: Large language models as hyper-heuristics with reflective evolution. *arXiv preprint arXiv:2402.01145*.
- Yu, Y.; Wu, M.; Lin, Y.; and Lobeckowski, N. G. 2025. THINK: Can Large Language Models Think-aloud? *arXiv preprint arXiv:2505.20184*.
- Zhao, J.; and Cheong, K. H. 2025. Visual Evolutionary Optimization on Combinatorial Problems with Multimodal Large Language Models: A Case Study of Influence Maximization. *arXiv preprint arXiv:2505.06850*.
- Zheng, Y.; Xing, Z.; Zhang, Q.; Jin, B.; Li, P.; Zheng, Y.; Xia, Z.; Zhan, K.; Lang, X.; Chen, Y.; et al. 2024. Planagent: A multi-modal large language agent for closed-loop vehicle motion planning. *arXiv preprint arXiv:2406.01587*.

A Discussion and Limitation

A.1 Outlook of MLES for Policy Discovery

This paper provides a preliminary demonstration of the Multimodal Large Language Model-assisted Evolutionary Search (MLES) framework’s ability to autonomously discover programmatic policies. As detailed in Appendix F, MLES successfully constructed a data processing pipeline for image data in the Car Racing task, guiding control decisions for steering, throttle, and braking with flawless performance across various tracks. These achievements highlight the promising potential of MLES for future advancements.

Modular feature processing and decision-making As observed in Appendix F, the programmatic policy discovered by MLES is modular, comprising two main components: feature processing and decision-making. MLES currently utilizes libraries like OpenCV for image data processing, highlighting its ability to autonomously employ such tools for feature processing. This also demonstrates the promising potential for integrating more advanced feature processing modules, such as convolutional neural networks, to further enhance data handling. These advanced modules would then feed processed features into an interpretable control logic for decision-making. Unlike deep reinforcement learning (DRL), where feature processing and decision-making are both black-box and tightly coupled, MLES ensures that even if feature processing is a black box, the decision-making remains interpretable. This clear modularization enables MLES to tackle complex scenarios while maintaining fully verifiable and explainable control logic. Such capabilities are essential for tasks like embodied intelligence and autonomous driving, where high levels of safety, transparency, and interpretability are critical, while needing to handle highly complex environments.

Moreover, this modular characteristic facilitates human expert involvement in the policy discovery process. In DRL, the tight coupling of the entire network makes it challenging to modify specific control preferences. MLES, however, allows experts to inspect and intervene at any stage of the policy discovery process. If a deficiency is identified, the expert can modify the corresponding part of the policy and reintroduce it into the policy pool. This enables continuous expert-assisted refinement. A further advantage is that experts can simply identify deficiencies and suggest directions for improvement in a prompt, allowing MLES to autonomously implement targeted optimizations.

Easier knowledge transfer and reuse A key advantage of MLES is that the discovered policies are encoded as executable code, which greatly simplifies knowledge transfer and reuse. In DRL, transferring knowledge often involves complex mechanisms like transfer learning or fine-tuning, which can be resource-intensive and difficult to implement. MLES bypasses these challenges by directly encoding prior knowledge into code. This approach not only makes knowledge transfer easier but also accelerates the policy deployment process. Additionally, LLMs can assist in this translation process, further reducing the time and effort required. As a result, MLES is particularly beneficial for tackling problems that involve well-established expert knowledge.

Moreover, this ease of transfer extends to addressing generalization challenges. In many tasks, different instances of the same problem may require subtle adjustments to the policy for optimal performance. Despite these variations, the core logic of the policy generally remains consistent. As such, instead of starting from scratch for each new task, MLES can adapt and fine-tune existing policies with minimal adjustments, ensuring continued high performance. A promising future direction for MLES is the collaborative evolution of policies for instances across multiple distributions, facilitating knowledge sharing of high-performing policies across different scenarios during the evolutionary process. This approach will be demonstrated in future work.

With the integration of visual feedback-driven behavior analysis, MLES mimics how human experts design policies—actively evaluating and adjusting policies in response to feedback. The key advantage is that MLES operates fully autonomously and can scale indefinitely, given sufficient computational resources. This scalability significantly accelerates the discovery of viable policies. In summary, MLES holds substantial potential to be a leading framework for interpretable policy discovery. We believe that MLES offers a new path forward for the field and invite the broader research community to explore and contribute to this exciting area.

A.2 Limitation

Construction of interpretable behavioral evidence MLES relies on behavioral evidence to analyze policy behavior and guide targeted improvements. This requires the behavioral evidence to be sufficiently informative, capturing key performance aspects. In this paper, for the Lunar Lander and Car Racing tasks, we manually designed interpretable behavioral evidence from a human perspective and successfully applied it within MLES. However, for new tasks, human experts need to design task-specific evidence construction processes, introducing a certain amount of manual design work. To address this, we propose two potential solutions. First, adopting video-based behavioral evidence could provide rich, detailed information with minimal manual effort. However, this approach is computationally expensive and would require a reduction in MLLM costs before it becomes widely applicable. Second, extending MLES to support the automatic design of behavioral evidence. Prior to policy discovery, MLES could automatically design evidence construction pipelines, thereby generating evidence that facilitates the understanding of the task and the policy’s behavior by MLLMs, thus reducing the manual effort required.

Dependency on multimodal large language models. The performance of MLES’s policy discovery process is inherently tied to the capabilities of the underlying MLLMs. Specifically, the analysis of behavioral evidence relies on the MLLMs’ visual understanding, while policy generation depends on their knowledge base and programming capabilities. As MLLMs continue to evolve, we anticipate that improvements in their capabilities will lead to enhanced policy discovery in MLES, allowing it to scale to more complex tasks and improve the overall effectiveness of the framework.

Higher cost compared to traditional deep reinforcement learning. Another limitation of MLES is its higher computational cost due to the need to interact with MLLMs. While visual feedback-driven behavior analysis improves the policy discovery efficiency, it also increases the cost per policy generation. To mitigate this, we suggest expressing richer information within a single image (as demonstrated in this paper), thereby reducing the number of tokens used and lowering computational costs. Moreover, as MLLM technology advances and costs decrease, the overall cost of policy design within MLES will be significantly reduced.

B Details of Quantitative Metrics in MLES

B.1 Quantitative Metrics for Policy Evaluation in MLES

In MLES, quantitative metrics serve as indicators of policy performance, analogous to the episode reward function in DRL. These metrics provide a consistent basis for ranking and selecting candidates during the policy evolution process. Unlike DRL, where each action is evaluated with immediate rewards in an online fashion, MLES evaluates the overall performance of a policy across a set of instances. In other words, MLES utilizes sparse rewards to guide policy learning, focusing on the final performance outcome.

This approach offers a practical advantage in that it eliminates the need for frequent manual design of intermediate rewards. We only need to focus on transforming the ultimate task goal into an evaluable metric, which can then support automated policy discovery. This not only simplifies the process but also alleviates the computational burden associated with reward design. However, this comes with the challenge of lacking direct evaluation of intermediate behaviors, which may potentially lead to issues such as reward hacking. To address this limitation, MLES incorporates *interpretable behavioral evidence* (IBE), which supplements the evaluation of intermediate actions. A more detailed discussion of this can be found in Section 3.3 and Appendix C.

Given a set of training instances, MLES aims to maximize (or minimize) the quantitative metric across the policies in its policy pool. During the evaluation process, the performance of a policy is typically assessed in parallel across all relevant instances, and the overall quantitative metric is computed as a weighted average of the metrics for each instance. Through iterative optimization, this process generates policies that are well-suited to instances drawn from the same distribution as the training instances.

B.2 Quantitative Metrics for Benchmark Tasks

This subsection outlines the quantitative metrics employed for evaluating policies in the Lunar Lander and Car Racing tasks.

Lunar Lander The quantitative metric for evaluating performance in the Lunar Lander task is the Normalized Weight Score (NWS), which is defined as follows:

$$\text{NWS} = \frac{R}{200} \times 0.6 + (1 - \min(\frac{C}{100}, 1)) \times 0.2 + S \times 0.2 \quad (3)$$

where R is the mean episode reward, C is the mean fuel consumption, and S is the success rate of completing the landing task, all computed over multiple instances.

For each training instance, the episode reward is calculated by summing the rewards across all steps taken by the policy during execution. The reward for each step is determined by the following criteria:

- Reward increases/decreases based on the proximity to the landing pad.
- Reward increases/decreases based on the lander’s speed (slower movement is rewarded).
- Reward decreases based on the tilt of the lander (less tilt is better).
- An additional reward of 10 points is awarded for each leg in contact with the ground.
- Penalties are imposed for using side or main engines during flight (-0.03 and -0.3 points per frame, respectively).
- A penalty of -100 points is incurred for crashing, and a bonus of +100 points is awarded for a successful landing.

A policy is considered successful if it achieves an episode reward of at least 200, which contributes to the first term in the NWS calculation. The policy’s success is also determined by its ability to land safely, which is reflected in the third term of the NWS. A policy with an NWS greater than 1 indicates a 100% safe landing rate, with larger values indicating more fuel-efficient landings. To ensure fair comparison, DRL algorithms also adopt the default reward function as described above.

Car Racing Regarding the Car Racing task, the Gym environment provides a reward of -0.1 points for each frame, with an additional reward of +1000/N for each track tile visited, where N is the total number of tiles in the track. This reward function incentivizes agents to complete the race track efficiently while avoiding driving off the course.

For the Car Racing task, we intuitively use the track completion rate as the quantitative metric for each instance. MLES aims to maximize the mean track completion rate across the training instances as its objective for policy discovery. In our experiment, DRL adopts the default reward function from the Gym environment, as described above.

C The Formats and Necessity of Interpretable Behavior Evidence

This section provides a detailed discussion of the potential formats of Interpretable Behavior Evidence (IBE) and the construction techniques involved, along with the specific forms adopted in this work and the rationale behind our choices.

C.1 Formats of Interpretable Behavior Evidence

The primary objective of IBE is to represent the behavior patterns associated with a given policy, thereby helping MLLMs in identifying policy shortcomings and facilitating subsequent improvements. IBE serves as a crucial supplement to the evaluation of intermediate actions, addressing the inherent limitations of purely quantitative metrics such as episode reward and success rate, which often fail to provide granular insights into policy execution.

The precise format of IBE is secondary to the quality and interpretability of the information it conveys, and how effectively this information can be leveraged by MLLMs. MLLM training paradigms, relying on human-provided images and corresponding natural language descriptions, progressively align model capabilities with human understanding. As evidenced by related work, current MLLMs demonstrate a level of visual comprehension closely mirroring human perception, enabling them to interpret information in a semantically meaningful way.

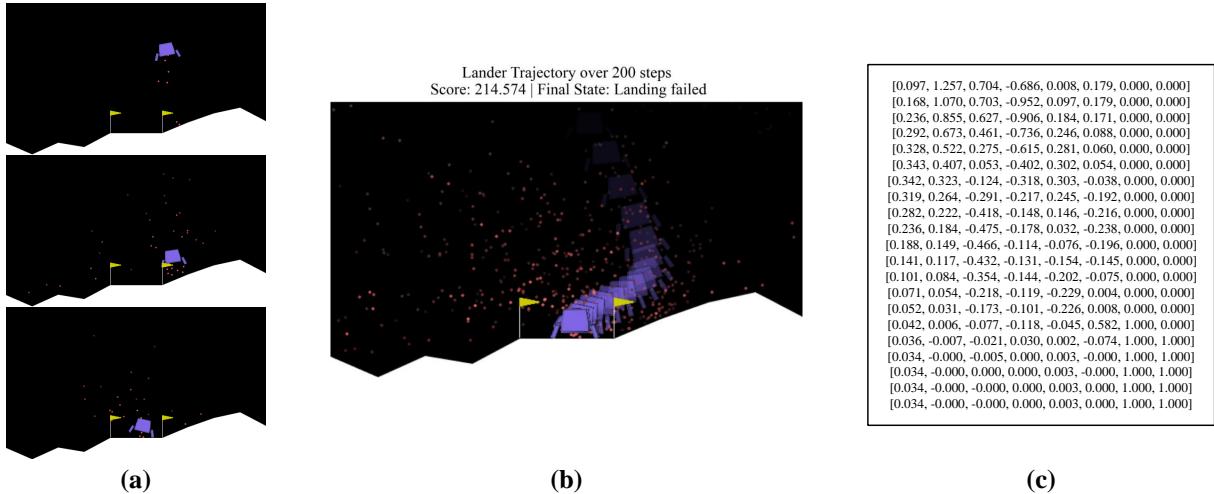


Figure 6: Examples of IBE in the Lunar Lander task. (a) Trajectory videos visualizing the agent’s motion over time. (b) Pose overlays summarizing sequential states in a single image. (c) State traces presenting a symbolic representation of the agent’s behavior.

Figure 6 illustrates three potential IBE formats within the context of the Lunar Lander task: videos, images, and textual state traces. While these formats convey the same underlying information, their interpretability and computational overhead for MLLM input vary significantly. Videos offer a complete visualization of the entire process but incur substantial computational expense. With appropriate preprocessing, images can effectively convey the control process while maintaining relatively low computational costs. Conversely, textual descriptions, though offering the lowest computational cost, are less intuitive for both human and MLLM interpretation, demanding greater cognitive effort for comprehension.

It is important to note that agent behaviors in numerous tasks are challenging to represent clearly using purely textual formats. For instance, in the Car Racing task within our benchmarks, coordinate-based trajectories or performance records in textual form fail to convey the intricate and subtle interactions between the car and the track. Such representations do not capture crucial nuances of how the agent controls its motion and the corresponding dynamic outcomes, which are vital for a comprehensive policy analysis. In contrast, visual representations offer significantly richer expressiveness for these dynamics, thereby facilitating better policy understanding by humans. As previously discussed, MLLMs can also leverage visual data to interpret and analyze the policy’s control process, thereby underpinning the visual feedback-driven behavior analysis within the MLES.

Furthermore, real-world control tasks, such as bipedal robot walking, drone stabilization, or autonomous driving, frequently involve complex dynamics that are inherently difficult to articulate or fully capture through textual means. Visual formats are therefore superior for conveying these complex behaviors. In our experiments, to strike a balance between comprehensibility for both humans and MLLMs and computational efficiency, we opt to employ images as the primary format for IBE. The specific methodologies for constructing them are detailed in the subsequent section.

C.2 IBE for Two Benchmarks

This section details the construction of IBE for Lunar Lander and Car Racing tasks, addressing their unique characteristics.

Lunar Lander. The Lunar Lander task features a static background, with only the lander object in motion. Given this setup, we can effectively capture the entire behavioral pattern by using a frame stacking technique with transparency. As illustrated in Fig. 7, this process involves selecting frames at fixed intervals and applying a transparency effect to each one before stacking them. The resulting IBE is a single image that visualizes the lander’s pose and trajectory at key moments, effectively compressing the time-series information of an entire episode into a compact, visual representation.

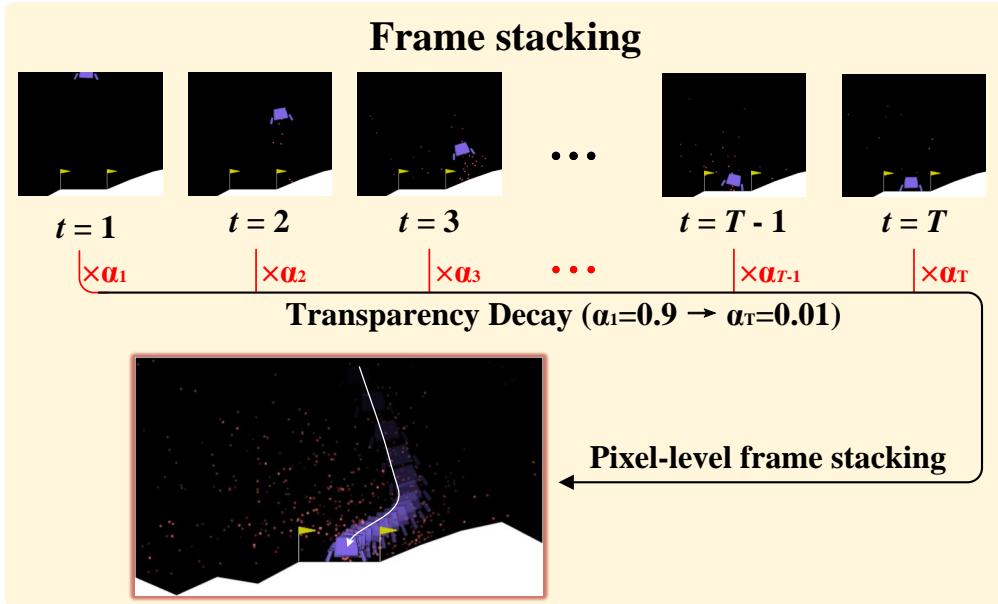


Figure 7: Frame stacking for IBE construction for the Lunar Lander task

Car Racing. The Car Racing task is a top-down environment where the camera continuously follows the car, and the background is dynamic. For this reason, the frame stacking method is not applicable. Instead, we propose an IBE construction method based on a global map and trajectory mapping, as shown in Fig. 8. The process begins by obtaining the complete coordinates of the track from the environment’s backend to construct a global track map. The car’s movement trajectory is then recorded throughout the episode and plotted onto this map upon completion. However, a simple trajectory alone is insufficient for explaining the policy’s decision-making process.

To address this limitation, we annotate the agent’s dynamic visual field at fixed environment step intervals along the trajectory. This enhancement provides two key benefits: (1) The density of the visual fields directly reflects the car’s speed: a denser distribution indicates a slower speed, and vice versa. (2) These visual fields provide crucial context, enabling MLLM to analyze the specific conditions that led to suboptimal actions, such as veering off the track. This targeted contextual information facilitates more precise reasoning and informed policy improvements.

C.3 Necessity of IBE

This section presents a qualitative analysis of IBE’s necessity, highlighting how it addresses a key limitation of quantitative metrics: their inability to evaluate the intermediate actions.

Fig. 9 illustrates the performance of policies designed by EoH and MLES on the same Car Racing track. Both policies achieve a perfect score, completing 100% of the track. However, a closer inspection of their trajectories reveals a critical difference. The EoH policy’s trajectory deviates significantly from the track on the final turn, only completing the course by executing a large corrective maneuver. This behavior is a classic example of “reward hacking,” a common issue in RL where agents exploit

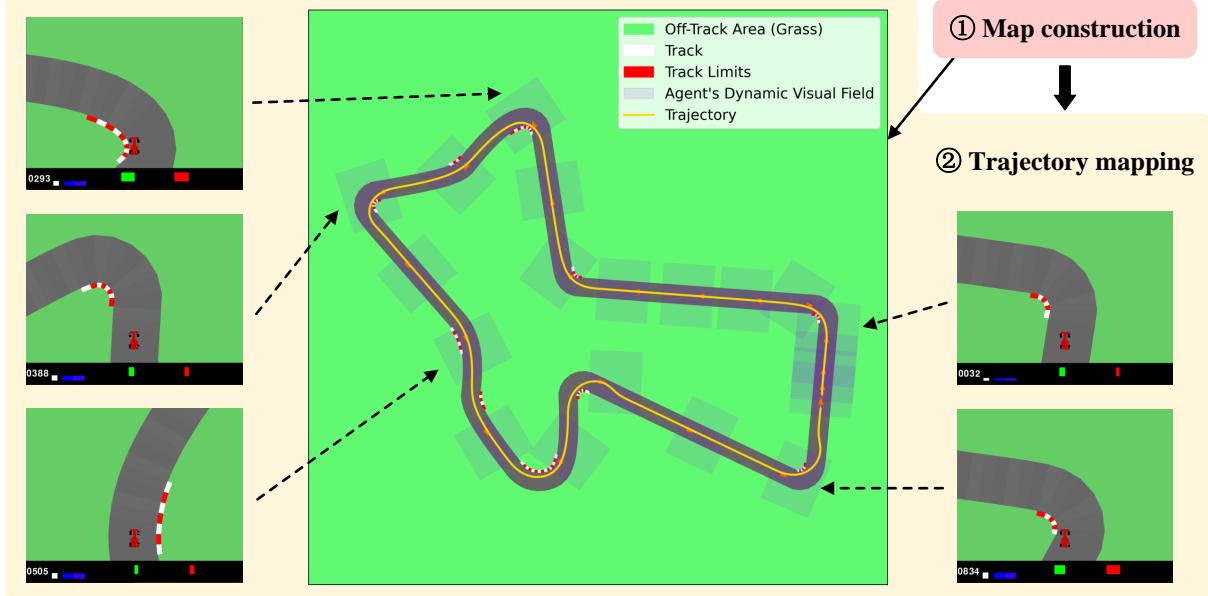


Figure 8: Trajectory mapping for IBE construction for the Car racing task

flawed reward functions to achieve a high score through suboptimal or unintended behaviors (Skalse et al. 2022). In contrast, the MLES policy’s trajectory is both rational and efficient, closely mirroring a human driver’s racing line.

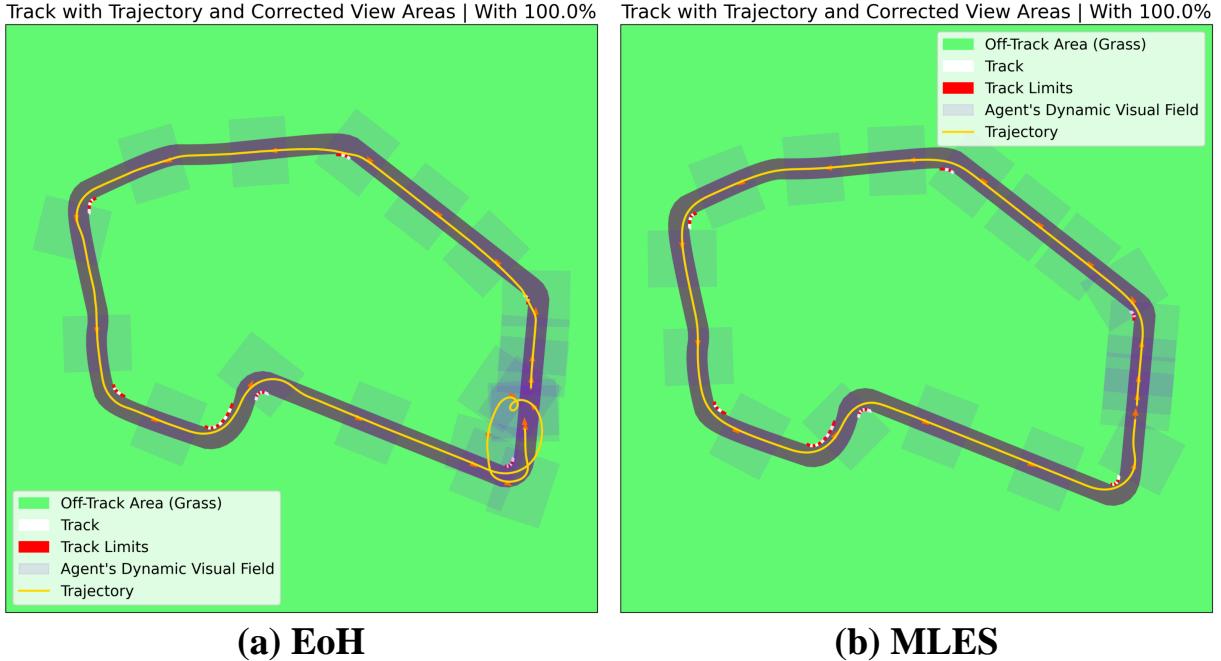


Figure 9: Trajectories of policies generated by EoH and MLES on a specific track, both achieving a perfect score

Because EoH’s evolutionary process is driven solely by a quantitative performance metric, it fails to evaluate the quality of the policy’s intermediate actions and thus cannot identify and correct such superficially high-performing but ultimately flawed policies. This leads to two significant problems: (1) Deceptively high-scoring policies can persist within the population, hindering the overall evolutionary progress. (2) Policies may achieve strong results on training instances but exhibit poor generalization to new, unseen instances.

In our approach, MLES utilizes IBE as a crucial supplement to quantitative metrics, allowing MLLMs to perform in-depth

behavioral analysis. This qualitative input enables the system to identify and proactively rectify these "fake-excellent" policies. Furthermore, for policies that are sound but underperforming, IBE allows the MLLM to provide targeted feedback for specific improvements, thereby accelerating the policy discovery process. This highlights the indispensable role of IBE in the LES paradigm for achieving robust and generalizable policy learning.

D Additional experiment results

D.1 Hyperparameters for Experimental Setup

This subsection presents the hyperparameters used in our experiments involving MLES, EoH, DQN, and PPO. The experiments were implemented in Python and executed on a single CPU (Intel i9-13980HX) with 32GB of RAM. Table 3 details the hyperparameters for MLES and EoH, where identical settings were applied to ensure a fair comparison, based on the configurations from the EoH paper (2024a). Tables 4 and 5 present the hyperparameter settings for DQN and PPO across two tasks, respectively. Tables 6 and 7 show the network architectures used for DQN and PPO on two benchmarks. To ensure a fair comparison, we maintained identical network structures (with approximately equal parameter counts) for DQN and PPO in terms of feature processing and decision-making, excluding the output layers.

Hyperparameter	Parameter Description	Value
m	Number of parents selected in each evolution step	2
N	Population size	16
LLM	Version of LLM used in the evolutionary operators	GPT-4o-mini
LLM temperature	Hyperparameter controlling the randomness of LLM text generation	1

Table 3: Hyperparameter settings for MLES and EoH.

Hyperparameter	Lunar Lander	Car Racing
Action Space Dimension	4	9 (Discrete actions)
Replay Buffer Capacity	10,000	30,000
Batch Size	128	128
Discount Factor	0.99	0.98
Initial ϵ	1.0	1.0
Minimum ϵ	0.01	0.1
ϵ Decay Rate	0.9995	0.995
Target Network Update Rate (τ)	0.01	0.01
Optimizer Type	Adam	Adam
Learning Rate	0.001	0.0005

Table 4: DQN Hyperparameters for Lunar Lander and Car Racing Tasks

Parameter	Lunar Lander	Car Racing
Discount Factor	0.99	0.99
Generalized Advantage Estimation Parameter	0.95	0.95
PPO Clip	0.2	0.2
Entropy Coefficient	0.01	0.01
Critic Coefficient	0.5	0.5
Max Gradient Norm	0.5	0.5
Batch Size	256	256
Mini Batch Size	64	64
Learning Rate	3e-5	3e-5

Table 5: PPO Hyperparameters for Lunar Lander and Car Racing Tasks

D.2 Statistical Analysis of Comparative Evaluations

Table 8 presents the average, standard error of the mean (SEM), and best results across five runs on training instances, complementing the data shown in Table 1. With the aid of visual feedback-driven behavior analysis, MLES demonstrates superior

Component	Layer Type	DQN Configuration	PPO Configuration
Input Layer	State Dimension	8 (environment observation space)	
Hidden Layer	Layer 1 Layer 2	Linear(512)→ReLU Linear(512)→ReLU	
Output Layer	Policy Head Value Head	Linear(4) N/A	Linear(1)
Activation	Hidden Layers Output Layer	ReLU Linear	
Parameters	Total Parameters	269,316	269,829

Table 6: Network Architectures for DQN and PPO on Lunar Lander Task

Component	Specification	DQN	PPO
Input Layer	Dimensions	96×96×3 (Current frame) + 96×96×3 (Last frame) + 1 (Speed) + 3 (Last action)	
Convolutional Layers	Layer 1	Conv2d(3→32, kernel=8×8, stride=4)→ReLU	
	Layer 2	Conv2d(32→64, kernel=4×4, stride=2)→ReLU	
	Layer 3	Conv2d(64→64, kernel=3×3, stride=1)→ReLU	
Shared Network	Architecture	Linear(8196→512)→ReLU	Linear(8196→512)→ReLU
Policy Head	Architecture	Linear(512→9)	Linear(512→256)→ReLU→Linear(256→3)
	Output	Discrete actions (9)	Continuous actions (steering, throttle, brake)
Value Head	Architecture	N/A	Linear(512→256)→ReLU→Linear(256→1)
Activation	Hidden	ReLU	
	Output	Linear	Policy: Tanh/Sigmoid, Value: Linear
Parameters	Total	4,277,417	4,536,484

Table 7: Network Architectures for DQN and PPO on Car Racing Task

stability and convergence compared to EoH. In the Car Racing task, both MLES and PPO are able to design policies that achieved a perfect score across all 10 test instances. Notably, as shown in Table 7, while PPO utilizes a significantly larger number of parameters, the policies designed by MLES involve considerably less computational complexity, making them more suitable for deployment in real-world applications.

Method	Lunar Lander			Car Racing		
	Ave	SEM	Best	Ave	SEM	Best
DQN	1.017	± 0.022	1.066	79.777	± 2.635	91.182
PPO	1.032	± 0.026	1.076	99.212	± 0.390	100.000
EoH	1.053	± 0.021	1.085	89.808	± 1.553	96.675
MLES	1.090	± 0.005	1.098	98.704	± 0.719	100.000

Table 8: Average, best results, and SEM from five runs on training instances

D.3 Influence of Behavioral Evidence and Prompt Engineering on Policy Discovery

This section investigates the influence of different forms of IBE and prompt engineering strategies on the policy discovery process. We employ M1 as the test operator and construct various few-shot prompts, incorporating diverse forms of IBE and specific instructions. Specifically, we evaluate the following five configurations:

- **M1**: The baseline operator derived from EoH, without any additional input.
- **M1_M**: Incorporates image inputs. The prompt explicitly instructs the MLLM to provide a detailed description and analysis of the image content.

	M1	M1_M	M1_M [†]	M1_T	M1_M [‡]
Population improvement	+19.43%	+24.55%	+20.21%	+20.29%	+19.73%
Best policy improvement	+1.18%	+4.39%	+2.37%	+3.64%	+2.95%

Table 9: Average performance improvement in population and best policy across different configurations for the M1 operator.

- **M1_M[†]:** Includes image input, but without explicit instructions for description or analysis. The LLM directly optimizes the policy based on the provided image.
- **M1_T:** Introduces textual trajectory coordinates, as illustrated in Fig. 6(c). The prompt explicitly instructs the MLLM to describe and analyze this information in detail.
- **M1_M[‡]:** Employs a two-stage process. The image is first processed by an MLLM 1 to generate a detailed textual description, which is then passed to MLLM 2 for M1-level reasoning and policy generation. This configuration is akin to using an LLM that does not have direct access to the image.

To quantitatively assess the impact of these different IBE forms and prompt instructions, we initialized our experiments with five distinct populations, each exhibiting significant evolutionary potential. These populations are carefully selected from an established policy evolution process for the Lunar Lander task by MLES. Specifically, we identify five generations that have previously demonstrated substantial progress and select their immediate preceding populations as our initial populations. This selection criterion ensures that our starting points possess considerable room for improvement, allowing for clearer observation of the operators’ effects. Each M1 operator variant is applied to these populations for two generations. The entire experiment is repeated five times to ensure statistical robustness. We measure the impact by observing both the average performance improvement of the population and the improvement of the best policy within each population.

Table 9 summarizes the average improvements in population performance and best policy performance brought about by these operators. Several key observations and insights can be drawn from these results:

1. Integrating IBE significantly boosts policy discovery efficiency. A foundational finding is that the incorporation of IBE consistently leads to improved performance compared to the baseline M1 operator. The baseline M1, without any IBE, achieves a population improvement of +19.43% and a best policy improvement of +1.18%. In contrast, all configurations that utilize IBE (M1_M, M1_M[†], M1_T, M1_M[‡]) exhibit higher gains in both population and best policy performance. For instance, M1_M demonstrates the highest population improvement at +24.55% and also the most significant best policy improvement at +4.39%. This clearly indicates that providing MLLMs with relevant behavioral evidence significantly aids in fine-grained policy evaluation and enables more targeted policy optimization. The evidence acts as a crucial guide, steering the MLLM toward more effective policy spaces.

2. The impact of interpretability and richness of IBE. Both M1_M and M1_T demonstrate substantial improvements in best policy performance, with +4.39% and +3.64%, respectively. While both forms of IBE prove beneficial, the visual representation of the lander’s behavior and posture in images is arguably more expressive, intuitive, and easier for the MLLM to interpret than raw textual state traces. This superior interpretability and rich expressiveness enable the MLLM to analyze policy behavioral patterns more deeply. These results support our claim, as discussed in Appendix C.1, that the most crucial aspect of IBE is the quality and inherent interpretability of the information it conveys, along with how effectively this information can be leveraged by MLLMs for comprehensive behavioral analysis and subsequent policy refinement. The richer, more direct semantics embedded in visual data appear to offer a distinct advantage.

3. The benefit of direct image input. Comparing M1_M (direct image input) with M1_M[‡] (two-stage processing where an image is first described by an MLLM into text, then passed to the policy generator), we observe a clear performance disparity. M1_M achieves a population improvement of +24.55% and a best policy improvement of +4.39%, whereas M1_M[‡] yields slightly lower gains at +19.73% for population and +2.95% for best policy. When an image transforms into a textual description by another model, even if that description is “detailed,” there is an inherent risk of information loss or misinterpretation. For an MLLM serving as a policy generator, directly accessing the raw image appears more advantageous than relying on a cascaded interpretation. This suggests that the rich, nuanced information present in the original image may be difficult to fully capture and convey through an intermediate textual representation, at least with current LLM capabilities. This outcome highlights the value of direct multimodal processing for inherently multimodal tasks, as it avoids potential bottlenecks and accumulated errors introduced by an intermediate textualization step.

4. The advantage of explicit instructions for behavioral analysis. The contrast between M1_M and M1_M[†] strongly highlights the advantage of providing explicit instructions for behavior analysis within the prompt. M1_M consistently performs better in both population (+24.55%) and best policy (+4.39%) improvement compared to M1_M[†] (+20.21% and +2.37% respectively). Both configurations receive image input, but M1_M explicitly instructs the LLM to provide a detailed description and analysis of the image, while M1_M[†] does not. This gap emphasizes that while MLLMs can implicitly derive information

from raw image input to aid inference, explicit directives to analyze IBE can provide rich, relevant context. This promotes the activation of pertinent knowledge within the MLLM. By encouraging a "thinking aloud" process, such instructions enable the LLM to synthesize more comprehensive and actionable insights, which are subsequently utilized to generate more effective policy modifications. This finding is consistent with observations in Chain of Thought and similar works (Yu et al. 2025; Wang and Venkatesh 2025).

In summary, these experiments underscore that both the form of IBE and the strategy of prompt engineering are crucial factors in maximizing the effectiveness of MLES-driven policy discovery. A well-crafted context, combining expressive and interpretable IBE with clearly instructed analytical processes during MLLM inference, significantly improves reasoning and enables more targeted and effective policy optimization. Our findings suggest that for complex tasks requiring a nuanced understanding of behavior, direct multimodal input, coupled with explicit analytical prompts, represents a superior approach.

E Prompts

E.1 Templates for Constructing Prompts for Specific Operators

This section details the prompt templates utilized for policy generation within the MLES framework. Figures 10, 11, 12, and 13 illustrate the specific prompt templates for operators E1, E2, M1_M, and M2_M, respectively. In all presented templates, black text indicates fixed components, red placeholders denote task-specific elements, and blue placeholders represent variables that evolve throughout the evolutionary process.

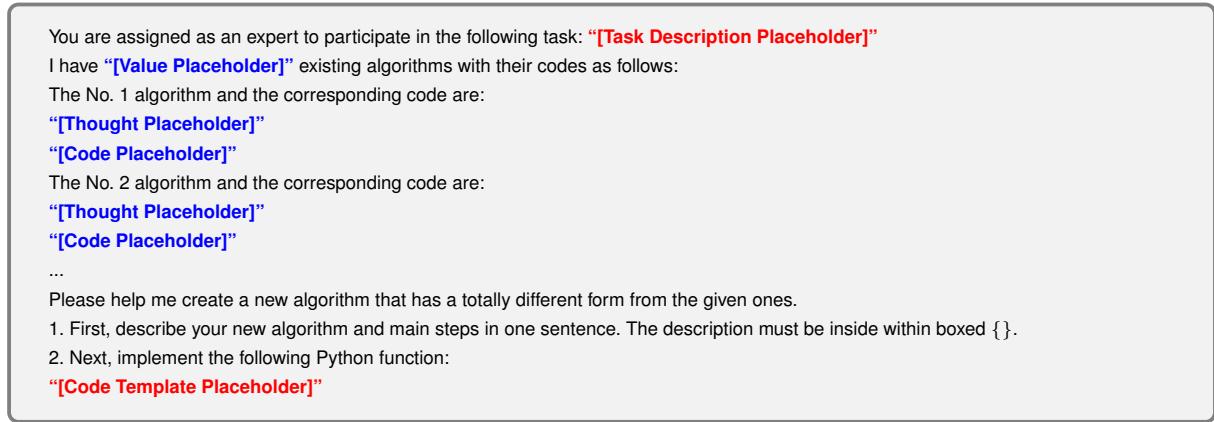


Figure 10: Prompt template for the E1 operator.

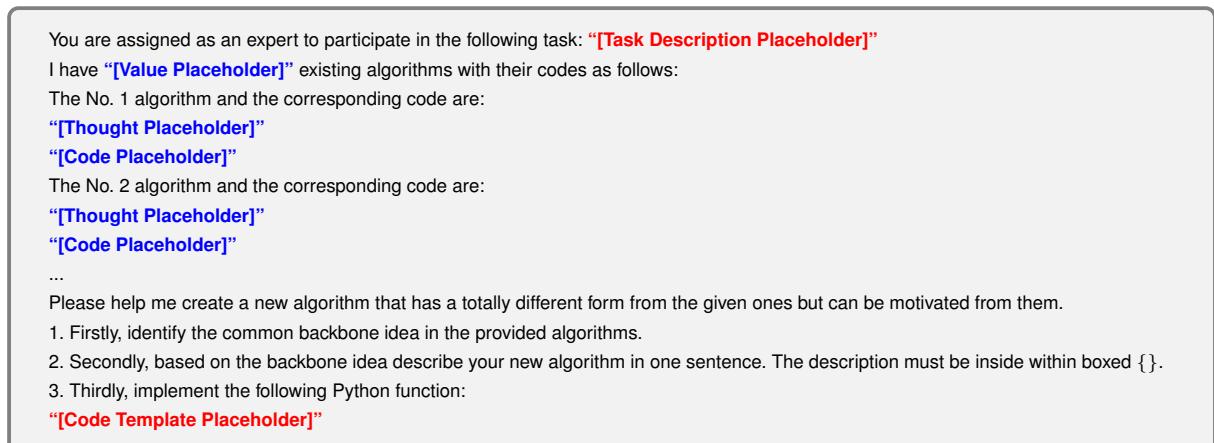


Figure 11: Prompt template for the E2 operator.

You are assigned as an expert to participate in the following task: “[Task Description Placeholder]”

We have a working algorithm that needs optimization. Below are its concept, implementation, and execution results:

Concept: “[Thought Placeholder]”

Implementation: “[Code Placeholder]”

Execution results visualization for the algorithm: “[Interpretable Behavioral Evidence Placeholder]”

Please start by providing a detailed description and analysis of the execution result, enclosed within single quotes (‘ ’). Next, based on your analysis, optimize the algorithm by following these steps:

1. Analyze why the results were produced in relation to the algorithm. Identify its weaknesses and areas for improvement, and enclose your analysis within square brackets [].
2. Propose an enhanced algorithm. Use concise language to describe the core idea of your algorithm, and enclose the core idea within curly braces {}.
3. Implement the enhanced algorithm using the following Python function template:
“[Code Template Placeholder]”

Figure 12: Prompt template for the M1_M operator.

You are assigned as an expert to participate in the following task: “[Task Description Placeholder]”

We have a working algorithm that needs optimization. Below are its concept, implementation, and execution results:

Concept: “[Thought Placeholder]”

Implementation: “[Code Placeholder]”

Execution results visualization for the algorithm: “[Interpretable Behavioral Evidence Placeholder]”

Please start by providing a detailed description and analysis of the execution result, enclosed within single quotes (‘ ’). Next, based on your analysis, optimize the algorithm by following these steps:

1. Parameter Analysis:
 - Identify all key parameters and their functions.
 - Determine which parameters should be modified to improve results.
 - Explain why these specific changes would help.
 - All content related to Parameter Analysis must be enclosed within brackets [].
2. Create a new algorithm that has a different parameter settings of the algorithm provided. Use concise language to describe the core idea of your algorithm, and enclose the core idea within curly braces {}.
3. Implement the enhanced algorithm using the following Python function template:
“[Code Template Placeholder]”

Figure 13: Prompt template for the M2_M operator.

E.2 Task-Specific Prompts Used in Experiments

This section demonstrates the design of task descriptions and code templates for applying MLES, using two control problems from our experiments as examples. Briefly, the task description communicates the requirements for the control policy to be designed, while the code template defines the programmatic policy's code interface that MLES needs to generate. Figures 14 and 15 present the task descriptions provided to MLES for the Lunar Lander and Car Racing tasks, respectively. Figures 16 and 17 show the corresponding code templates.

Task description. The task description serves as a problem statement for MLES, akin to an assignment given by a teacher to a student. First, we establish the overall context of the policy discovery task by clearly stating the objective: what control policy MLES needs to design. This primes the MLLMs to inherently prepare relevant knowledge for policy generation. Second, we explicitly articulate the desired characteristics of an optimal policy, guiding the LLM's value alignment with human intent. Furthermore, we can provide a more detailed description of the control task to facilitate precise programming by MLES. For example, in Fig. 15, we elaborate on observation details, such as the type of observation data and key visual elements. More comprehensive information tends to activate more relevant knowledge within the LLM.

Implement a novel heuristic strategy function that guides the lander in selecting actions step-by-step to achieve a safe landing. At each step, an appropriate action could be chosen based on the lander's current state and previous state, with the objective of reaching the target location in as few steps as possible. A 'safe landing' is defined as a touchdown with low vertical speed, upright orientation, and both angular velocity and angle close to zero, and both legs in contact with the ground.

Figure 14: Task description for the Lunar Lander task.

Write a Python function that serves as a control strategy for an agent in a top-down car racing environment.

Environment Overview

In this environment, the agent is required to drive a car along a race track. The primary objective is to cover as much of the track surface as possible before the time limit expires. To accomplish this, the agent needs to efficiently navigate the track by controlling the car's steering, throttle, and brake.

Observation Details

The agent's observation consists of a $96 \times 96 \times 3$ RGB image representing the top-down view of the environment. The following key visual elements can be identified in this image:

- Car: Red (approximately $[\approx 202, < 10, < 10]$).
- Track: Gray (approximately $[\approx 102, \approx 102, \approx 102]$).
- Off-track grass: Greenish (approximately $[\approx 102, \approx 204, \approx 102]$).
- Curbs (Sharp Turns): High-contrast red and white (approximately $[> 240, < 20, < 20]$ and $[> 240, > 240, > 240]$).

Inputs at Each Time Step

At every time step, the agent receives the following information:

- The current RGB observation of the environment.
- The current speed of the car.
- The previous RGB observation and the previous action taken by the agent.

Function Requirements

The Python function should incorporate a control policy that combines visual perception from the RGB observations and past information (previous observation and action). This policy should enable the agent to maintain optimal control of the car, keep the car on the track, and maximize the efficiency of lap completion.

Figure 15: Task description for the Car Racing task.

Code Template. The code template defines the communication protocol and can optionally furnish MLES with expert knowledge. A code template typically consists of three parts: function declaration (function name and parameters), docstring, and function body.

Within the function declaration and docstring, we must pre-define the parameters that the policy and environment will exchange, ensuring compatibility with the environment's interface. This involves: (1) identifying the parameters the environment or system provide to the policy and defining them as input variables in the function declaration; (2) detailing the type, dimension, range, and meaning of these input parameters in the docstring; and (3) clearly specifying the policy's output in the docstring, typically the actions the agent can take. Additionally, we can incorporate prior knowledge and hints within the docstring to

enhance the effectiveness of policies generated by MLES. As shown in Figure 17, we can include "Notes" sections within the docstring to help MLES better understand the environment and offer specific guidance. For the function body, we can simply provide a return statement to ensure function completeness, or we can additionally include existing expert heuristics. Generally, providing some expert knowledge facilitates the policy discovery process of MLES.

```

1 import numpy as np
2
3 def choose_action(s: list, last_action: int, s_pre: list) -> int:
4     """
5         Selects an action for the Lunar Lander to achieve a safe landing at the target location (0, 0).
6         Args:
7             s (list or np.ndarray): The current state of the lander. Elements:
8                 s[0] - horizontal position (x)
9                 s[1] - vertical position (y)
10                s[2] - horizontal velocity (v_x)
11                s[3] - vertical velocity (v_y)
12                s[4] - angle (radians)
13                s[5] - angular velocity
14                s[6] - 1 if the first leg is in contact with the ground, else 0
15                s[7] - 1 if the second leg is in contact with the ground, else 0
16            last_action (int): The action taken in the previous step. One of:
17                0 - do nothing
18                1 - fire left orientation engine
19                2 - fire main (upward) engine
20                3 - fire right orientation engine
21            s_pre (list or np.ndarray): The state of the lander *before* the last action was executed.
22        Returns:
23            int: The chosen action for the next step. One of:
24                0 - do nothing
25                1 - fire left orientation engine
26                2 - fire main (upward) engine
27                3 - fire right orientation engine
28        """
29        angle_targ = s[0] * 0.5 + s[2] * 1.0 # angle should point towards center
30        if angle_targ > 0.4:
31            angle_targ = 0.4 # more than 0.4 radians (22 degrees) is bad
32        if angle_targ < -0.4:
33            angle_targ = -0.4
34        hover_targ = 0.55 * np.abs(
35            s[0]
36        ) # target y should be proportional to horizontal offset
37
38        angle_todo = (angle_targ - s[4]) * 0.5 - (s[5]) * 1.0
39        hover_todo = (hover_targ - s[1]) * 0.5 - (s[3]) * 0.5
40
41        if s[6] or s[7]: # legs have contact
42            angle_todo = 0
43            hover_todo = (
44                -(s[3]) * 0.5
45            ) # override to reduce fall speed, that's all we need after contact
46
47        a = 0
48        if hover_todo > np.abs(angle_todo) and hover_todo > 0.05:
49            a = 2
50        elif angle_todo < -0.05:
51            a = 3
52        elif angle_todo > +0.05:
53            a = 1
54    return a

```

Figure 16: Code template for the Lunar lander Task

```

1 import numpy as np
2 import cv2
3 def choose_action(observation, car_speed, pre_action, pre_observation):
4     """
5         Determine the next action for the Car Racing agent.
6         This function takes into account the current state (observation and speed), the previous action, and the previous
7         observation.
8         Notes:
9             - The car in this environment is a powerful rear-wheel-drive vehicle. Avoid accelerating while turning sharply,
10                as this can easily lead to loss of control.
11            - Occasionally, track segments (e.g., after a U-turn) may appear in the observation but are not part of the
12                immediate drivable path. These should be distinguished to avoid premature or incorrect decisions.
13            - Avoid coming to a complete stop, as this may prevent the car from finishing the race.
14        Args:
15            observation (np.ndarray): The current state observed by the agent, represented as a 96x96 RGB image of the
16                car and race track from a top-down view (shape: (96, 96, 3)).
17            car_speed (float): The current speed of the car.
18            pre_action (np.ndarray): The action taken by the car in the previous step, represented as a 3-element array.
19            pre_observation (np.ndarray): The observation received when the previous action was taken. It has the same
20                shape and format as 'observation' (i.e., a 96x96 RGB image).
21        Returns:
22            np.ndarray: The action selected by the agent for the next step, represented as an array of shape (3,) where:
23                - Index 0: Steering, where -1 is full left, +1 is full right (range: [-1, 1]).
24                - Index 1: Gas, (range: [0, 1]).
25                - Index 2: Braking, (range: [0, 1]).
26
27
28    # Create 3D gray detection mask (all RGB channels within range)
29    gray_mask = (
30        (observation[:, :, 0] >= gray_low) & (observation[:, :, 0] <= gray_high) &
31        (observation[:, :, 1] >= gray_low) & (observation[:, :, 1] <= gray_high) &
32        (observation[:, :, 2] >= gray_low) & (observation[:, :, 2] <= gray_high)
33    )
34
35    gray_indices = np.argwhere(gray_mask)
36    center_x = np.mean(gray_indices[:, 1]) if len(gray_indices) > 0 else observation.shape[1] // 2
37    car_position = observation.shape[1] // 2
38    offset = center_x - car_position
39
40    steering_angle = np.clip(offset / 100.0, -1.0, 1.0)
41    action[0] = steering_angle
42
43    if abs(offset) > 10:
44        action[1] = 0.0
45        action[2] = 0.2
46    else:
47        action[1] = 0.8
48        action[2] = 0.0
49
50    gray_density = np.sum(gray_mask) / (observation.shape[0] * observation.shape[1])
51    if gray_density < 0.1:
52        action[1] = 0.4
53        action[2] = 0.3
54
55    return action

```

Figure 17: Code template for the Car Racing Task

F Generated Programmatic Policy

This section presents the best policies discovered by MLES, including the code, thought, and performance on a range of instances.

Lunar Lander. Fig. 18 shows the code for the best policy discovered for the Lunar Lander task. The thought of this policy provided by MLES is: "The core idea of the new algorithm is to further increase the sensitivity to vertical velocity, making the lander more responsive to abrupt changes, while also lowering the action thresholds to enable quicker decisions for stability control." Fig.19 displays the performance of this policy across several instances, highlighting its ability to achieve safe landings consistently.

```

1 def choose_action(s: list, last_action: int, s_pre: list) -> int:
2     """
3         Selects an action for the Lunar Lander to achieve a safe landing at the target location (0, 0).
4         Args:
5             s (list or np.ndarray): The current state of the lander. Elements:
6                 s[0] - horizontal position (x)
7                 s[1] - vertical position (y)
8                 s[2] - horizontal velocity (v_x)
9                 s[3] - vertical velocity (v_y)
10                s[4] - angle (radians)
11                s[5] - angular velocity
12                s[6] - 1 if the first leg is in contact with the ground, else 0
13                s[7] - 1 if the second leg is in contact with the ground, else 0
14            last_action (int): The action taken in the previous step. One of:
15                0 - do nothing
16                1 - fire left orientation engine
17                2 - fire main (upward) engine
18                3 - fire right orientation engine
19            s_pre (list or np.ndarray): The state of the lander *before* the last action was executed.
20        Returns:
21            int: The chosen action for the next step. One of:
22                0 - do nothing
23                1 - fire left orientation engine
24                2 - fire main (upward) engine
25                3 - fire right orientation engine
26        """
27        angle_targ = (s[0] * 0.5 + s[2] * 0.5) # Designed for better orientation control
28        angle_targ = np.clip(angle_targ, -0.6, 0.6) # Slightly wider bounds for orientation
29
30        # Update hover target with amplified sensitivity to vertical speed
31        hover_targ = 0.3 * np.abs(s[0]) + 0.7 * (s[3] ** 2) # Greater weight for hover stabilizing
32
33        # Calculate actions for adjustments
34        angle_todo = (angle_targ - s[4]) * 1.6 - (s[5]) * 1.2 # More aggressive response for angle adjustments
35        hover_todo = (hover_targ - s[1]) * 1.3 - (s[3]) * 0.4 # Enhanced weight for vertical control
36
37        # Stabilizing when legs are in contact
38        if s[6] or s[7]:
39            angle_todo = 0 # Maintain upright position priority
40            hover_todo -= (s[3]) * 0.4 # Stronger adjustment for vertical speed reduction
41
42        # Decision-making based on refined thresholds and more responsive measures
43        a = 0
44        if hover_todo > np.abs(angle_todo) and hover_todo > 0.10: # Slightly tighter hover threshold
45            a = 2 # Fire main engine
46        elif angle_todo < -0.15: # Increased sensitivity for left engine
47            a = 3
48        elif angle_todo > 0.15: # Increased sensitivity for right engine
49            a = 1
50    return a

```

Figure 18: Code of the best discovered policy for the Lunar Lander task.

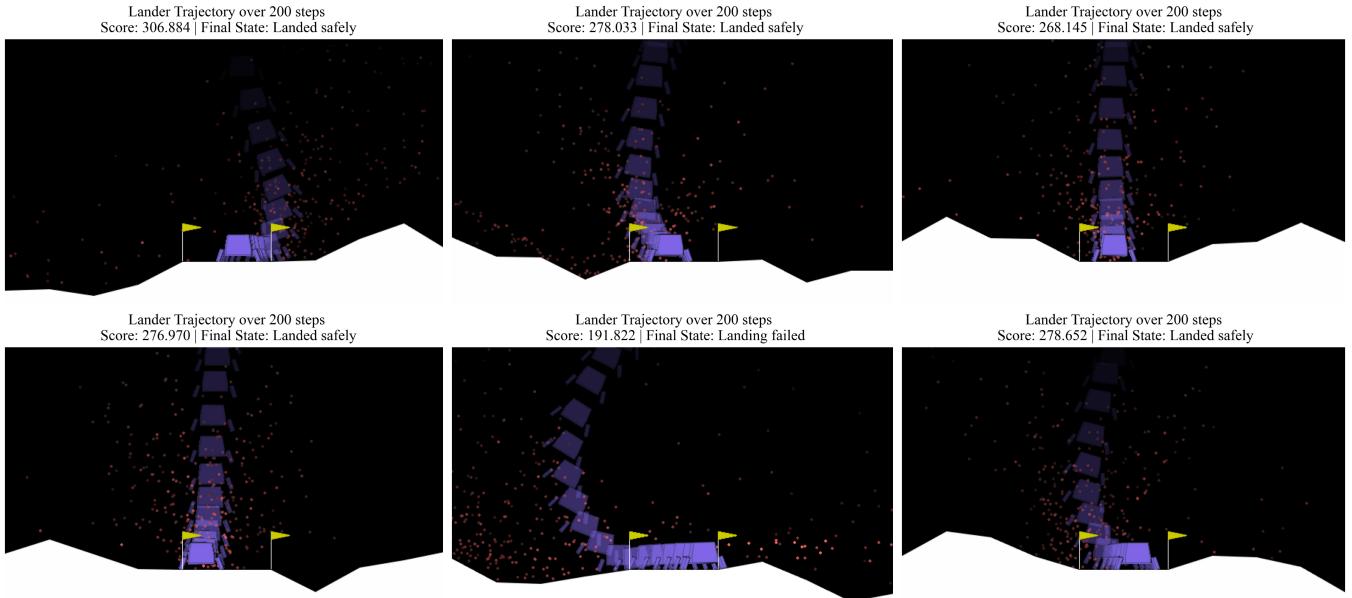


Figure 19: Performance of the best discovered policy on different instances in the Lunar Lander task. It shows the lander's successful landings and stability under various conditions.

Car Racing. Fig. 20 shows the code for the best policy discovered for the Car Racing task. The core idea of this policy is to dynamically adjust the steering sensitivity and throttle response based on the car's speed and previous actions. This adjustment enables smoother control over the car's acceleration and deceleration, ensuring optimal performance during races. Furthermore, the policy incorporates an improved method for assessing track boundaries, focusing on smoother transitions between acceleration and deceleration. The policy intelligently uses the car's speed to modulate steering and throttle, allowing for better control in turns while avoiding overly sharp maneuvers that could destabilize the vehicle. Fig. 21 illustrates the policy's performance on several instances, demonstrating its ability to complete the race smoothly, with racing lines that resemble those of human drivers.

```

1 def choose_action(observation, car_speed, pre_action, pre_observation):
2     """
3         Determine the next action for the Car Racing agent.
4         Args:
5             observation (np.ndarray): The current state observed by the agent, represented as a 96x96 RGB image of the
6                 car and race track from a top-down view (shape: (96, 96, 3)).
7             car_speed (float): The current speed of the car.
8             pre_action (np.ndarray): Action taken by the agent in the previous step, represented as a 3-element array.
9             pre_observation (np.ndarray): The observation received when the previous action was taken. It has the same
10                shape and format as 'observation' (i.e., a 96x96 RGB image).
11        Returns:
12            np.ndarray: The action selected by the agent for the next step, represented as an array of shape (3,) where:
13                - Index 0: Steering, where -1 is full left, +1 is full right (range: [-1, 1]).
14                - Index 1: Gas, (range: [0, 1]).
15                - Index 2: Braking, (range: [0, 1]).
16        """
17     hsv = cv2.cvtColor(observation, cv2.COLOR_RGB2HSV)
18     # Define masks for track, off-track, and curbs detection
19     track_mask = cv2.inRange(hsv, (0, 0, 60), (179, 255, 200)) # Track pixels
20     off_track_mask = cv2.inRange(hsv, (40, 100, 100), (90, 255, 255)) # Off-track pixels
21     curbs_mask = cv2.inRange(hsv, (180, 100, 100), (255, 255, 255)) # Curbs detection
22
23     # Analyze the center region for better decision making
24     height, width = track_mask.shape
25     center_region = track_mask[int(height / 3):int(2 * height / 3), int(width / 4):int(3 * width / 4)]
26     # Initialize the action array
27     action = np.zeros(3)
28     # Identify whether track is present and where
29     track_detected = np.sum(center_region) > 0
30     if not track_detected:
31         action[2] = 1.0 # Full brake if no track detected
32     else:
33         # Identify track pixel positions
34         track_pixels = np.column_stack(np.where(center_region > 0))
35         mean_x = np.mean(track_pixels[:, 1]) if len(track_pixels) > 0 else center_region.shape[1] / 2
36
37         # Steering adjustment with consideration for immediate curvature
38         steering = (mean_x - (center_region.shape[1] / 2)) / (center_region.shape[1] / 2)
39         action[0] = np.clip(steering, -1, 1) # Normalize steering value
40
41     # Adaptive throttle management based on speed
42     if car_speed < 1.0:
43         action[1] = 1.0 # Full throttle if the car is very slow
44     else:
45         # Adjust throttle considering steering
46         if abs(action[0]) > 0.5:
47             action[1] = max(0.0, 1.0 - abs(action[0])) # Reduce throttle during significant turns
48         else:
49             action[1] = min(1.0, action[1] + 0.05) # Gradual increase on straights
50
51     # Enhanced braking logic considering sharp turns
52     if abs(action[0]) > 0.6 and car_speed > 2.0:
53         action[2] = min(0.4, action[2] + 0.1) # Moderate braking for sudden turns
54     else:
55         action[2] = 0.0 # No brake needed on straights
56
57     # Prevent the car from stalling
58     if np.all(pre_action == [0, 0, 1]) and car_speed < 0.5:
59         action[1] = max(action[1], 0.5) # Ensure some throttle to avoid stalling
60
61     return action

```

Figure 20: Code of the best discovered policy for the Car Racing task.

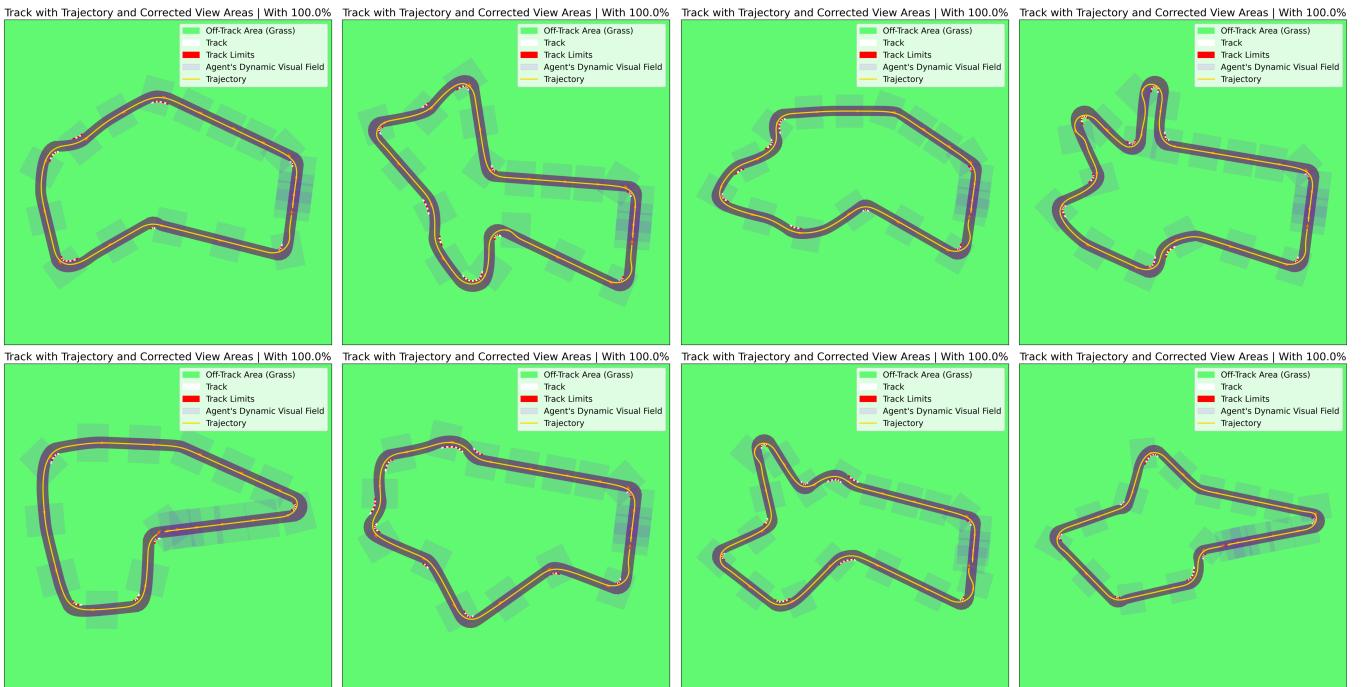


Figure 21: Performance of the best discovered policy on different instances in the Car Racing task. These figure shows the car’s smooth track completion and race line strategy, similar to human racing behavior.