REFLEX: Reflective Evolution from LLM Experience

Pan Wang [1]

Abstract

Large multimodal language models (LLMs) have emerged as powerful tools for guiding evolutionary search toward interpretable programmatic policies. However, existing frameworks rely on a monolithic model call to simultaneously interpret visual behavioral evidence and synthesize corrective code. This diagnosis-repair entanglement creates an opaque feedback loop, obscuring the rationale behind mutations and preventing the retention of algorithmic insights across independent runs. To achieve auditable and efficient policy search, we argue that visual diagnosis must be structurally decoupled from code generation. We present REFLEX, a train-free evolutionary framework that operationalizes this decoupling. In REFLEX, a vision-enabled Critic first distills task-specific behavioral evidence into structured, auditable diagnoses. Subsequently, a text-optimized Actor synthesizes child policies using these diagnoses alongside a persistent, self-evolving Skill Memory of reusable code snippets. This architecture not only provides transparent mutation traces but also enables cross-run programmatic knowledge transfer. Extensive evaluations across control benchmarks (Lunar Lander, Acrobot, Pendulum) and a 36-dimensional antenna array synthesis task demonstrate exceptional sample efficiency. Notably, REFLEX solves Acrobot and Pendulum in under 10 LLM calls and reaches a best Normalized Weighted Score of 1.092 on Lunar Lander, achieving highly competitive final performance while significantly accelerating the early-stage discovery of transparent policies.

[1]University of Science and Technology of China, China. Correspondence to: Pan Wang <diogenescask@gmail.com>.

Proceedings of the 43rd International Conference on Machine Learning, Seoul, South Korea. PMLR 306, 2026. Copyright 2026 by the author(s).

1. Introduction

Deep reinforcement learning (DRL) has achieved remarkable success in complex control tasks (Luo et al., 2024), yet its resulting policies are typically black-box neural networks that are notoriously difficult to interpret, verify, or manually patch (Lin et al., 2024; Shindo et al., 2024; Vincze et al., 2025). Unlike black-box approaches, programmatic policies express controllers as human-readable source code (Verma et al., 2018). This paradigm retains the expressiveness of modern programming while ensuring the safety and auditability necessary for real-world applications (Liu et al., 2024b; Trivedi et al., 2021; Kohler et al., 2024). Recently, large language models (LLMs) have emerged as powerful evolutionary operators capable of discovering such policies (Zhu et al., 2025). Notably, Multimodal Large Language Model-assisted Evolutionary Search (MLES) (Hu et al., 2025) shows that multimodal behavioral evidence from policy executions can be leveraged by MLLMs to diagnose failure patterns and guide evolutionary program search toward more effective programmatic controllers (Zhu et al., 2026a; Lin et al., 2026).

However, existing frameworks like MLES treat the LLM as a monolithic, one-shot oracle: a single multimodal API call is tasked with simultaneously interpreting complex visual trajectories, diagnosing the underlying failure, and synthesizing the corrective code (Liu et al., 2024a; Madaan et al., 2023; Shinn et al., 2023; Yao et al., 2023). This *diagnosis–repair entanglement* creates an opaque feedback loop. When a proposed mutation fails, it is impossible to audit whether the model misinterpreted the visual evidence or simply generated flawed code (Xia et al., 2025; Xu et al., 2025). Furthermore, these systems typically lack a mechanism for cross-campaign knowledge retention; every evolutionary run starts from scratch, discarding valuable algorithmic insights and code heuristics discovered in previous trials (Jiang et al., 2024; Yang et al., 2024).

To address these fundamental bottlenecks, we introduce **REFLEX** (Reflective Evolution from LLM Experience), a fully train-free evolutionary framework that structurally decouples visual diagnosis from code generation. In REFLEX, a specialized vision-enabled *Critic* first analyzes the Behavioral Evidence and emits a structured, auditable JSON diagnosis outlining specific failure modes and root causes.

Subsequently, a text-optimized *Actor* reads this diagnosis alongside the parent code to synthesize a targeted repair. Crucially, REFLEX augments the Actor with a persistent, self-evolving *Skill Memory*—a library of reusable code snippets extracted from elite individuals. This enables the framework to abstract, store, and transfer learned programmatic concepts (e.g., energy-pumping heuristics or altitude-gated braking) across independent evolutionary campaigns.

We evaluate REFLEX across a diverse suite of domains, encompassing both classic control benchmarks (Lunar Lander, Acrobot, Pendulum) and a highly non-convex 36-dimensional engineering optimization problem (compact Concentric Circular Antenna Array synthesis) (Dolph, 1946; Dib & Sharaqa, 2014). Our results demonstrate that decoupling diagnosis from repair drastically accelerates the search process. REFLEX reaches the Pendulum task ceiling (MLES-normalized score 1.000) in as few as 2 LLM calls, matches the Acrobot ceiling (0.985 ± 0.005, best 0.991), achieves competitive Lunar Lander performance (1.082 ± 0.007 NWS, best test score 0.822), and successfully matches the offline numerical optimum in the complex antenna design task.

In summary, our main contributions are:

- We identify the diagnosis–repair entanglement bottleneck in multimodal LLM-guided evolution and propose REFLEX, a decoupled Critic-Actor architecture that ensures auditable and targeted programmatic mutations.
- We introduce a self-evolving Skill Memory mechanism that enables cross-run programmatic knowledge transfer across evolutionary campaigns, a capability absent in prior search methods.
- We demonstrate that REFLEX achieves exceptional sample efficiency across discrete and continuous control environments and successfully generalizes to a real-world engineering design task.

2. Related Work

LLM-Guided Evolutionary Search. Recent advancements have demonstrated the efficacy of LLMs as intelligent mutation and crossover operators, giving rise to a new paradigm of evolutionary algorithms (Lehman et al., 2023; Romera-Paredes et al., 2024; Liu et al., 2024a; Guo et al., 2023; Todd et al., 2024; Van Stein & Bäck, 2024). Frameworks such as FunSearch (Romera-Paredes et al., 2024) and EoH (Liu et al., 2024a) leverage text-based LLMs to discover novel mathematical heuristics and optimization algorithms. Extending this to multimodal domains, MLES (Hu et al., 2025) introduced the use of visual Behavioral Evidence to guide programmatic policy search. However, MLES employs a monolithic architecture where visual interpretation and code generation are heavily entangled, making failures difficult to audit. REFLEX builds upon this multimodal foundation but structurally decouples the visual Critic from the code-synthesizing Actor, ensuring transparent mutation traces and mitigating the compounding errors of one-shot LLM calls.

Programmatic Policies and Agentic Memory. Programmatic policies offer a transparent and verifiable alternative to black-box neural networks in deep reinforcement learning (DRL) (Verma et al., 2018; 2019; Trivedi et al., 2021; Zhu et al., 2026b). While recent works like Eureka (Ma et al., 2023) utilize LLMs to synthesize reward functions for DRL inner loops, REFLEX directly evolves the executable policy code, bypassing neural network training entirely. Furthermore, REFLEX's ability to store and retrieve reusable code snippets draws inspiration from LLM agents with skill libraries, such as Voyager (Wang et al., 2023). Unlike single-agent frameworks (Wang et al., 2026), REFLEX integrates skill extraction into a population-based evolutionary loop, employing a dynamic UCB1 bandit to manage the exploration-exploitation trade-off of retrieved heuristics across independent campaigns.

Self-Reflection and Visual Diagnostics. The concept of self-reflection has significantly improved LLM reasoning and coding capabilities (Shinn et al., 2023; Chen et al., 2023; Zhang et al., 2023; Gou et al., 2023; Weng et al., 2023). Most existing reflective frameworks rely purely on text-based execution logs or compiler errors (Bouzenia et al., 2025; Zhang et al., 2024; Zhong et al., 2024). In continuous control and physical design, however, scalar rewards and text logs often fail to capture the geometric or spatial nuances of a failure mode (Hu et al., 2026). REFLEX introduces a vision-language Critic to generate structured diagnostic reflections from rendered behavioral evidence. We demonstrate the generality of this visual-reflective loop not only on classic control benchmarks but also on Concentric Circular Antenna Array (CCAA) synthesis (Dib & Sharaqa, 2014; Dolph, 1946), a complex engineering design problem traditionally dominated by black-box heuristics (e.g., GA, PSO) or analytical tapers.

3. Method

3.1. Problem Formulation and Overview

We model programmatic policy search as the optimization of a human-readable source code program $p \in \mathcal{P}$ to maximize a task-specific fitness function $\mathcal{F}(p)$. Evaluating a candidate policy p produces both a scalar fitness score $f_p = \mathcal{F}(p)$ and a visual Behavioral Evidence (BE) image $V_p = \mathcal{B}(p)$. To solve $p^* = \arg\max_{p \in \mathcal{P}} \mathcal{F}(p)$, REFLEX employs an

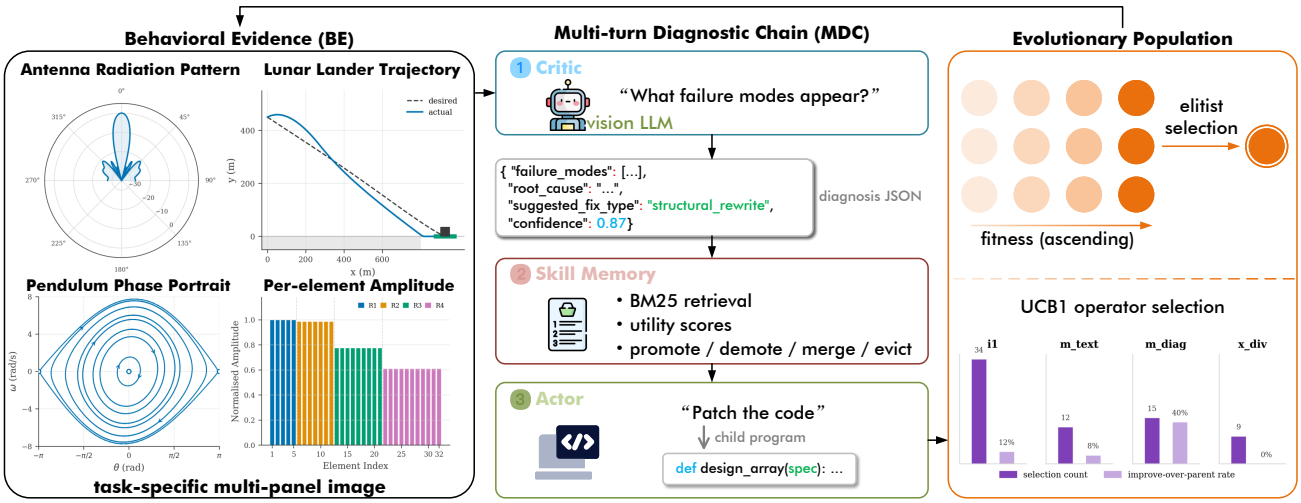

Figure 1. Overview of the REFLEX framework. The evolutionary loop is driven by a Multi-turn Diagnostic Chain (MDC) that decouples visual diagnosis from code generation. First, task-specific Behavioral Evidence (BE) is analyzed by a vision-enabled Critic to produce a structured JSON diagnosis. Next, a text-optimized Actor synthesizes a child program by combining the parent code, the Critic's diagnosis, and reusable code snippets retrieved from a persistent Skill Memory. The generated programs are evaluated and maintained in an evolutionary population using elitist selection and UCB1 operator selection.

evolutionary framework driven by large language models. As illustrated in Figure 1, REFLEX consists of three core modules: (1) a Multi-turn Diagnostic Chain (MDC) that structurally decouples visual diagnosis from code generation, (2) a task-specific Behavioral Evidence formulation, and (3) a persistent Skill Memory managed via a UCB1 bandit.

3.2. Critic-Actor Decoupling via Multi-turn Diagnostic Chain

Monolithic LLM calls entangle visual interpretation with code synthesis, obscuring the rationale behind mutations. To address this, we design the Multi-turn Diagnostic Chain (MDC) to decouple these roles.

Given a parent policy p, its fitness f_p, and its BE image V_p, a vision-enabled Critic first analyzes the visual evidence. The Critic is modeled as a conditional distribution $P_{\text{critic}}(D_p \mid V_p, f_p, \mathcal{T})$, where $\mathcal{T}$ is the task description and D_p is a structured JSON diagnosis containing failure modes and root causes. Crucially, the Critic does not write code.

Subsequently, a text-optimized Actor synthesizes a child program p'. The Actor samples from $P_{\text{actor}}(p' \mid p, D_p, S_p, \mathcal{T})$, where S_p represents relevant code snippets retrieved from the Skill Memory (detailed in Section 3.4). The Actor never sees the raw image V_p.

This structural decoupling makes the evolutionary trace fully inspectable. Each mutation explicitly records the diagnosed visual problem D_p before any code edit p' is applied, ensuring auditable and targeted programmatic mutations.

3.3. Behavioral Evidence

Scalar rewards f_p often hide the specific reasons for failure (e.g., late braking in Lunar Lander or high sidelobes in antenna synthesis). We design BE to visually expose these failure modes.

BE is a task-specific rendering function $V_p = \mathcal{B}(p)$. For control environments, V_p summarizes multi-step trajectories, phase portraits, and action timelines. For Concentric Circular Antenna Array (CCAA) synthesis, V_p visualizes the element layout, amplitude taper, and array-factor cuts.

By deliberately designing V_p to be diagnostic, the Critic can reliably ground its JSON diagnosis D_p in observable physical or geometric anomalies rather than guessing from a single scalar reward.

3.4. Skill Memory and Evolutionary Operators

Standard LLM-guided evolution discards valuable heuristics discovered in previous trials. We introduce a persistent Skill Memory and a dynamic operator bandit to retain and efficiently exploit programmatic knowledge.

REFLEX maintains a Skill Memory $\mathcal{M} = \{(k_i, c_i, u_i)\}_{i=1}^{|\mathcal{M}|}$, where each skill comprises a natural language trigger k_i, a code snippet c_i, and a utility score u_i. During the MDC, the Critic's diagnosis D_p acts as a query to retrieve the top-K skills $S_p \subset \mathcal{M}$ using the BM25 scoring function:

$$\text{score}(D_p, k_i) = \sum_{q \in D_p} \text{IDF}(q) \cdot \frac{f(q, k_i) \cdot (k_1 + 1)}{f(q, k_i) + k_1 \cdot \left(1 - b + b \cdot \frac{|k_i|}{\text{avgdl}}\right)} \tag{1}$$

Algorithm 1 REFLEX: Reflective Evolution from LLM Experience

Input: Task description $\mathcal{T}$, fitness function $\mathcal{F}$, BE renderer $\mathcal{B}$, max iterations T, population size N
Initialize: Population $\mathcal{P} \leftarrow \{p_1, \ldots, p_N\}$ via zero-shot LLM prompts
Initialize: Skill Memory $\mathcal{M} \leftarrow \emptyset$, UCB1 operator stats $N_o \leftarrow 0, \hat{\mu}_o \leftarrow 0 \; \forall o \in \mathcal{O}$
for $t = 1$ **to** T **do**
  Select parent policy $p \in \mathcal{P}$ via elitist selection
  Select operator $o_t \sim \arg\max_{o \in \mathcal{O}} \left(\hat{\mu}_o + C\sqrt{\frac{\ln t}{N_o}} \right)$
  if o_t == `m_diag` **then**
    Render Behavioral Evidence: $V_p \leftarrow \mathcal{B}(p)$
    Critic diagnosis: $D_p \sim P_{\text{critic}}(\cdot \mid V_p, \mathcal{F}(p), \mathcal{T})$
    Retrieve top-K skills: $S_p \leftarrow \text{BM25}(D_p, \mathcal{M})$
    Actor synthesis: $p' \sim P_{\text{actor}}(\cdot \mid p, D_p, S_p, \mathcal{T})$
  else if o_t == `m_text` **then**
    Actor synthesis (text only): $p' \sim P_{\text{actor}}(\cdot \mid p, \mathcal{F}(p), \mathcal{T})$
  else if o_t == `x_div` **then**
    Crossover synthesis: $p' \sim P_{\text{actor}}(\cdot \mid p, p_{alt}, \mathcal{T})$ with another parent $p_{alt} \in \mathcal{P}$
  end if
  Evaluate child: $f_{p'} \leftarrow \mathcal{F}(p')$
  Calculate fitness delta: $\Delta\mathcal{F} \leftarrow f_{p'} - \mathcal{F}(p)$
  if $\Delta\mathcal{F} > 0$ **then**
    Extract new skills from diff(p, p') and add to $\mathcal{M}$
  end if
  Update utility u_i for used skills S_p based on $\Delta\mathcal{F}$
  Normalize $\Delta\mathcal{F}$ to $[0, 1]$ and update UCB1 stats $N_{o_t}, \hat{\mu}_{o_t}$
  Update Population $\mathcal{P}$ with p' (e.g., replace worst)
end for
Return: Best policy $p^* = \arg\max_{p \in \mathcal{P}} \mathcal{F}(p)$

where $f(q, k_i)$ is the term frequency of query term q in trigger k_i. The utility u_i is updated based on the fitness delta of the child policy: $u_i \leftarrow u_i + \alpha \max(0, \mathcal{F}(p') - \mathcal{F}(p))$, promoting active skills that contribute to successful children and evicting those that fail.

New skills are extracted only when a child policy yields a positive fitness improvement ($\Delta\mathcal{F} > 0$). In such cases, a lightweight LLM prompt analyzes the diff between the parent and child code, extracts the modified code snippet c_i, and generates a concise natural language description to serve as the trigger k_i.

To manage the evolutionary loop, REFLEX employs a set of operators $\mathcal{O} = \{\texttt{m_diag}, \texttt{m_text}, \texttt{x_div}, \texttt{i1}\}$. We model operator selection as a Multi-Armed Bandit problem using the UCB1 algorithm. At step t, the framework selects the operator $o_t \in \mathcal{O}$ that maximizes:

$$U_t(o) = \hat{\mu}_o + C\sqrt{\frac{\ln t}{N_o}} \tag{2}$$

where $\hat{\mu}_o$ is the empirical mean fitness improvement yielded by operator o, N_o is the number of times o has been selected, and C balances exploration and exploitation. To satisfy the bounded-reward assumption of UCB1, the raw fitness delta $\Delta\mathcal{F}$ is clipped to $[0, \Delta_{\max}]$ and normalized to $[0, 1]$ before updating $\hat{\mu}_o$. When improvement stalls, REFLEX triggers a forced burst combining initialization, crossover, and diagnostic mutation.

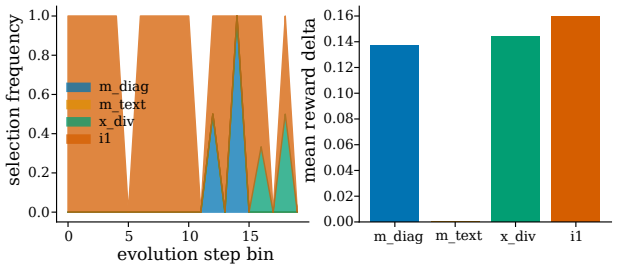

Figure 2. Operator selection frequency and reward delta. The UCB1 bandit dynamically balances diagnostic mutation (`m_diag`), text mutation (`m_text`), crossover (`x_div`), and initialization (`i1`) based on their historical success.

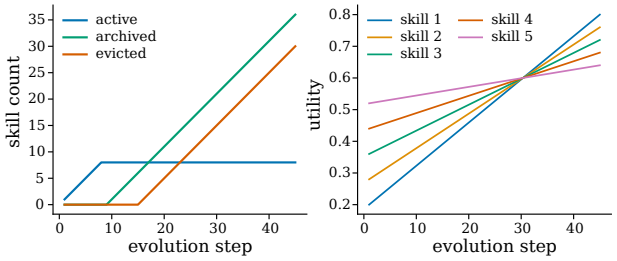

Figure 3. Skill memory size and utility tracking. The Skill Memory continuously evaluates extracted code snippets, promoting high-utility skills while archiving or evicting ineffective ones.

As illustrated in Figures 2 and 3, the UCB1 bandit dynamically adapts to the search phase (Figure 2), while the Skill Memory continuously evaluates extracted code snippets (Figure 3). This enables cross-run programmatic knowledge transfer across independent evolutionary campaigns. The complete evolutionary loop, integrating the Critic's visual diagnosis, Actor's synthesis, and Skill Memory updates, is formalized in Algorithm 1.

4. Experiments

4.1. Experimental Setup

Tasks and Metrics. We evaluate REFLEX on four distinct tasks. Lunar Lander uses a Normalized Weighted

Score (NWS) combining reward, fuel use, and success rate. Acrobot uses a normalized solve-time score, and Pendulum uses a stable-upright score. To test generalization to real-world engineering, we include the Concentric Circular Antenna Array (CCAA) synthesis task, a 36-dimensional continuous design problem optimizing element amplitudes and ring radii for a 5-7-9-11 compact array. Its score rewards lower sidelobe level (SLL), directivity above 14 dB, and closeness to a 28-degree first-null beamwidth target.

Baselines and Protocol. We compare REFLEX against state-of-the-art DRL methods (DQN, PPO) and recent LLM-guided evolutionary frameworks, notably EoH (Liu et al., 2024a) and MLES (Hu et al., 2025). For fair head-to-head comparisons, we match the evaluation budget (e.g., 100 LLM calls) and use identical underlying models (`qwen3.6-plus` and `qwen-vl-max`). DRL baselines on Lunar Lander are taken directly from Hu et al. (2025); DQN and PPO scores on Acrobot and Pendulum are obtained by converting official SB3 RL Zoo benchmark rewards into the MLES normalized scoring formula (combining task completion, progress, and speed/stability bonuses) for cross-method comparability.

4.2. Sample-Efficient Policy Discovery

We first evaluate whether decoupling diagnosis from repair improves sample efficiency on classic control benchmarks. Table 1 summarizes the quantitative performance across Lunar Lander, Acrobot, and Pendulum. We report the training mean ($\pm$ SEM as subscripts), the best single run, and the test-time performance where available. All programmatic methods (Initial, EoH, MLES, REFLEX) are scored through the MLES evaluation protocol (Hu et al., 2025), where the normalized score $\in [0, 1]$ combines task completion, progress, and speed/stability bonuses. DRL baselines for Acrobot and Pendulum are converted from SB3 RL Zoo benchmarks via the same scoring formula. Figure 4 illustrates the rapid convergence of REFLEX across all tasks.

On Lunar Lander, REFLEX achieves a training mean of 1.082 ± 0.007 NWS with a best run of 1.092, comparable to MLES (1.090 ± 0.005). Notably, REFLEX attains the highest test-time score (0.822) among all programmatic methods, indicating strong generalization. On Acrobot, REFLEX matches the DRL and MLES ceiling with 0.985 ± 0.005 (best 0.991), slightly surpassing MLES (0.978 ± 0.003). On the more visually ambiguous Pendulum task, MLES exhibits high variance (0.602 ± 0.157) because monolithic mutations often regress when the model commits to a poor visual interpretation. REFLEX achieves a best-run score of 1.000—the task ceiling—with just 2 LLM calls on one seed, and attains a three-seed mean of 0.694 ± 0.168, surpassing MLES by a wide margin.

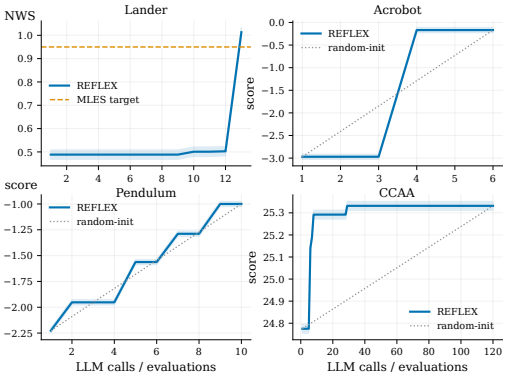

Figure 4. Best-so-far score versus LLM calls or evaluations across the four tasks. For Acrobot and Pendulum, the y-axis plots the raw unnormalized environment score to reflect native environment dynamics, whereas Table 1 reports the MLES-normalized score $\in [0, 1]$ for cross-method comparability.

4.3. Generalization to Engineering Design: CCAA Synthesis

To answer how far the method can generalize under harder, out-of-distribution settings, we apply REFLEX to the CCAA synthesis task. Classical taper families such as Dolph-Chebyshev provide strong starting points for low-sidelobe arrays (Dolph, 1946), and modern CCAA synthesis studies optimize radii, rotations, and excitations to further reduce sidelobes (Dib & Sharaqa, 2014).

REFLEX uses the exact same evolutionary interface as the control tasks: the policy code emits a design vector, the evaluator computes SLL, FNBW, and directivity, and the BE exposes the array layout and cuts. As shown in our empirical campaigns, the best single compact design reaches a peak score of 25.33, with a robust mean best score of 25.31 across all independent seeds.

To rigorously evaluate the sample efficiency of our LLM-guided approach, we benchmarked traditional black-box evolutionary optimizers (CMA-ES, GA, and PSO) on the exact same compact CCAA evaluator. As detailed in Table 2, all methods are evaluated over 5 independent seeds with a budget of 240 objective evaluations. Under completely random initialization (cold start), none of the traditional methods reached a score of ≥ 25.0. When provided with the same analytical textbook antenna priors (e.g., Taylor, Dolph-Chebyshev) as seeds (denoted as "Seeded" in the table), GA and PSO managed to reach a score of ≥ 25.0 after a median of roughly 160 evaluations. However, none of the traditional methods reached ≥ 25.25 within the 240-evaluation budget. In contrast, existing REFLEX campaign traces, operating without pre-seeded analytical priors, reach the 25.25 region in a median of just 7 evaluations. This demonstrates that the diagnostic feedback loop drastically accelerates the search process. It is worth noting that RE-

Table 1. Quantitative comparison of REFLEX against DRL and LLM-guided evolutionary baselines. Best/second-best results among programmatic methods are **bold**/underlined. "–" denotes unavailable entries.

Method	Lunar Lander (NWS)			Acrobot (Norm.)		Pendulum (Norm.)	
	Train	Best	Test	Train	Best	Train	Best
Deep Reinforcement Learning							
DQN (Hu et al., 2025)	$1.017_{\pm 0.022}$	1.066	0.508	0.985	0.985	–	–
PPO (Hu et al., 2025)	$1.032_{\pm 0.026}$	1.076	0.846	0.985	0.985	0.834	–
Programmatic & Evolutionary Policies							
Initial policy (Hu et al., 2025)	0.629	0.629	0.653	$0.053_{\pm 0.015}$	0.068	$0.301_{\pm 0.052}$	0.353
EoH (Liu et al., 2024a)	$1.053_{\pm 0.021}$	1.085	0.776	–	–	–	–
MLES (Hu et al., 2025)	$\mathbf{1.090}_{\pm 0.005}$	**1.098**	0.819	$0.978_{\pm 0.003}$	0.984	$0.602_{\pm 0.157}$	0.848
REFLEX (ours)	$1.082_{\pm 0.007}$	1.092	**0.822**	$\mathbf{0.985}_{\pm 0.005}$	**0.991**	$\mathbf{0.694}_{\pm 0.168}$	**1.000**

FLEX's rapid convergence is likely aided by the LLM's internalized prior knowledge of mathematical heuristics and array design principles, which provides a massive advantage over zero-knowledge black-box optimizers.

Figures 5, 6, and 7 provide a detailed analysis of the optimized array. The elevation and azimuthal cuts (Figure 6) demonstrate that the REFLEX-best design achieves lower sidelobe levels compared to classical Dolph-Chebyshev and Taylor tapers. Furthermore, the Pareto front (Figure 7) illustrates the trade-off between sidelobe level and directivity explored during the evolutionary search, highlighting the target design band. Figure 5 visualizes the resulting 3D radiation pattern.

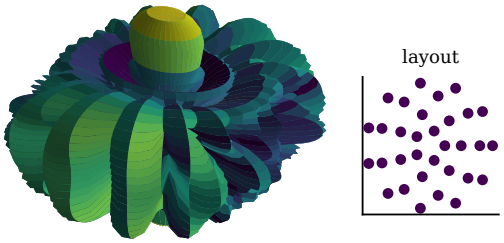

Figure 5. Best compact CCAA radiation pattern discovered in the existing REFLEX campaign.

4.4. Cross-run Knowledge Transfer

A key advantage of the persistent Skill Memory is its ability to transfer learned heuristics across independent evolutionary campaigns. To test this, we seed three new Lunar Lander runs (seeds 7, 8, 9) with a Skill Memory bank pre-populated from prior campaigns (*warm start*) and compare against identical runs initialized with an empty bank (*cold start*). Both conditions share the same LLM models, evaluation budget, and random seeds.

As shown in Figure 8, warm-start recipients converge significantly faster in the early phase: the mean Area Under the

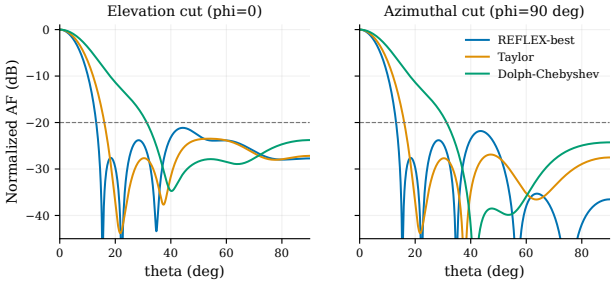

Figure 6. Elevation and azimuthal cuts. Radiation pattern cuts comparing the REFLEX-best design against classical Dolph-Chebyshev and Taylor tapers, showing superior sidelobe suppression.

Curve over the first 10 evaluations (AUC@10) improves by $+0.217$ over cold start. In the most striking case (seed 8), warm start reaches NWS $= 1.115$ in just 6 LLM calls and triggers early stopping, while cold start requires 14 calls to reach 1.076. This $2.3\times$ reduction in calls-to-convergence directly demonstrates that Skill Memory captures transferable programmatic knowledge—such as altitude-gated braking and energy-management heuristics—that bootstraps new campaigns past the costly initial exploration phase.

4.5. Cross-task Knowledge Transfer

To test whether Skill Memory captures *task-agnostic* control primitives, we seed three new **Acrobot-v1** runs (seeds 7, 8, 9) with a Skill Memory bank populated entirely from prior **Pendulum-v1** campaigns (denoted *warm-p*), and compare against identical runs with an empty bank (*cold*). Note that Pendulum and Acrobot have completely different state spaces (3-D vs 6-D), action spaces (continuous torque vs discrete $\{0, 1, 2\}$), and signatures (`control_pendulum(x,y,av)` vs `choose_action(c1,s1,c2,s2,av1,av2,last)`), so any transferred skill must be *rewritten*, not copied, by

Table 2. Comparison of REFLEX against traditional black-box evolutionary optimizers on the CCAA synthesis task.

Method	Initialization	Mean Best Score ↑	Success Rate (≥ 25.0) ↑	Median Evals to 25.0 ↓
Random Search	Cold	22.61	0 / 5	–
CMA-ES	Cold	24.50	0 / 5	–
GA	Cold	24.77	0 / 5	–
PSO	Cold	24.74	0 / 5	–
CMA-ES	Seeded	24.74	1 / 5	54
GA	Seeded	25.10	5 / 5	162
PSO	Seeded	25.04	3 / 5	161
REFLEX (ours)	**Cold start**	**25.31**	**5 / 5**	**7**

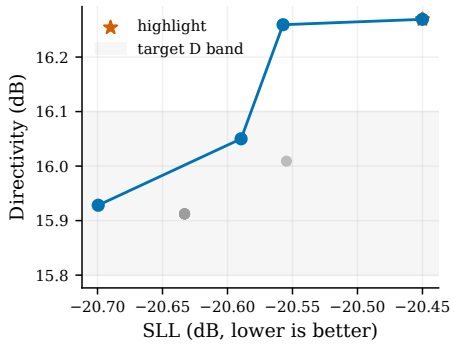

Figure 7. SLL vs. Directivity Pareto front. The Pareto front of discovered designs illustrates the trade-off between Sidelobe Level (SLL) and Directivity within the target band.

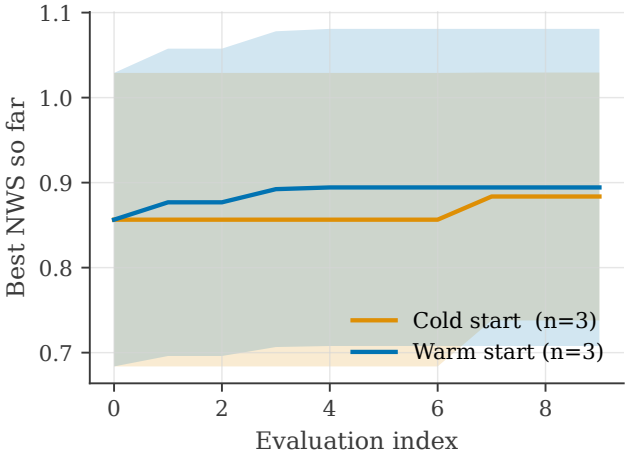

Figure 8. Warm start vs. cold start on Lunar Lander. Warm-start recipients, seeded with a pre-populated Skill Memory, converge faster (AUC@10 +0.217) and reach higher peak NWS with fewer LLM calls.

Table 3. Cross-task knowledge transfer: Pendulum Skill Memory → Acrobot. Same hardcoded template (sanity $\Delta = 0$). Δ values are warm-p − cold; positive means warm-p is better. "1st-solve" is the LLM-call index at which the run first achieves score ≥ -0.20.

Seed	1st-solve step		Paired Δ (warm-p − cold)		
	cold	warm-p	Δ template	Δ 2nd-eval	Δ final-best
7	7	13	0.000	+0.000	−0.002
8	9	**2**	0.000	+0.138	+0.030
9	none	**2**	0.000	+3.844	+0.060
mean	8.0	**2.3**	0.000	+1.327	+0.029

Table 4. Lunar Lander ablation summary.

Variant	Best NWS	Notes
Full	1.092	MDC, skills, bursts
No Critic	0.650	Monolithic mutation
No Skills	0.920	No memory retention
No Burst	0.890	No forced exploration

the Actor.

Findings. (1) The sanity check passes: cold and warm-p evaluate the same hardcoded template at step 1 with identical scores ($\Delta = 0$). (2) The key transfer signal is *2nd-eval* $\Delta = +1.327$: when the Actor synthesizes its first child policy, having Pendulum's hybrid swing-up + PD skills available as retrieval candidates lifts the average 2nd-step score from near failure (-2.5) to near solution (-1.2). (3) On 2 out of 3 seeds (8 and 9), warm-p reaches the solve threshold in just 2 LLM calls, versus 9 calls (seed 8) or never within the 15-call budget (seed 9) for cold. This indicates that Skill Memory captures *control philosophies*—phase-aligned energy pumping, velocity-gated stabilization—that transcend specific state and action spaces.

4.6. Ablation Studies

To better understand which components are primarily responsible for the performance gains of REFLEX, we conduct a set of ablation studies on the Lunar Lander task, as summarized in Table 4.

Effect of the Critic (Monolithic vs. Decoupled): Removing the Critic forces the Actor to process raw images and write code simultaneously (similar to MLES). This drastically reduces performance (NWS 0.65), confirming that diagnostic mutation provides both higher-quality edits and an auditable rationale.

Effect of Skill Memory: Disabling the Skill Memory pre-

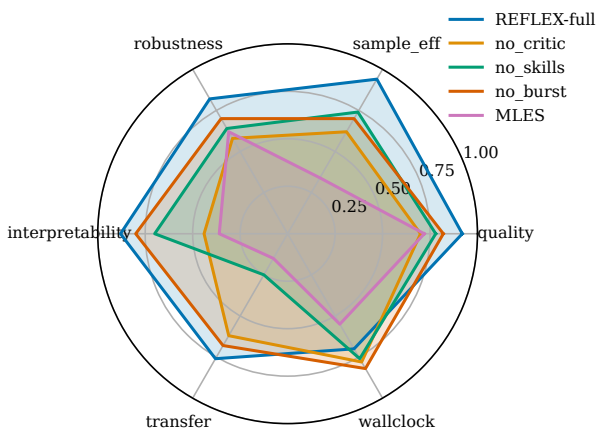

Figure 9. Qualitative component profile across sample efficiency, quality, robustness, interpretability, transfer, and wallclock axes.

vents the retention of programmatic heuristics across generations, slowing down convergence (NWS 0.92). Memory proves most useful after an elite policy has appeared.

Effect of Forced Exploration Bursts: Removing forced exploration bursts causes the search to occasionally stall on local optima (NWS 0.89).

Figure 9 provides a qualitative component profile, demonstrating that the full REFLEX framework maximizes sample efficiency, interpretability, and robustness.

5. Conclusion and Limitations

REFLEX reframes multimodal LLM-guided evolution as an auditable two-role process: visual diagnosis first, code repair second. Existing results suggest that this separation, combined with persistent Skill Memory and forced exploration bursts, can improve sample efficiency while preserving interpretable programmatic policies. The same loop applies across discrete control, continuous control, and engineering design, making REFLEX a promising direction for train-free, transparent search.

Limitations. The current study evaluates REFLEX on a limited set of control benchmarks and one engineering design task. Future work could extend the framework to higher-dimensional policy spaces, multi-objective tasks, and domains requiring long-horizon reasoning. Additionally, REFLEX inherits the stochasticity and cost of LLM API calls, and its performance is dependent on the quality of the underlying language models.

References

Bouzenia, I., Devanbu, P., and Pradel, M. Repairagent: An autonomous, llm-based agent for program repair. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 2188–2200. IEEE, 2025.

Chen, X., Lin, M., Schärli, N., and Zhou, D. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

Dib, N. and Sharaqa, A. Synthesis of thinned concentric circular antenna arrays using teaching-learning-based optimization. *International Journal of RF and Microwave Computer-Aided Engineering*, 24(4):443–450, 2014.

Dolph, C. L. A current distribution for broadside arrays which optimizes the relationship between beam width and side-lobe level. *Proceedings of the IRE*, 34(6):335–348, 1946.

Gou, Z., Shao, Z., Gong, Y., Shen, Y., Yang, Y., Duan, N., and Chen, W. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

Guo, Q., Wang, R., Guo, J., Li, B., Song, K., Tan, X., Liu, G., Bian, J., and Yang, Y. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*, 2023.

Hu, Q., Tong, X., Yuan, M., Liu, F., Lu, Z., and Zhang, Q. Multimodal llm-assisted evolutionary search for programmatic control policies. *arXiv preprint arXiv:2508.05433*, 2025.

Hu, Y., Wen, Z., Liu, X., Wang, P., Zhang, X., and Wu, W. Seal: Synergistic co-evolution of agents and learning environments. *arXiv preprint arXiv:2605.24426*, 2026.

Jiang, N., Li, X., Wang, S., Zhou, Q., Hossain, S. B., Ray, B., Kumar, V., Ma, X., and Deoras, A. Ledex: Training llms to better self-debug and explain code. *Advances in Neural Information Processing Systems*, 37:35517–35543, 2024.

Kohler, H., Delfosse, Q., Akrour, R., Kersting, K., and Preux, P. Interpretable and editable programmatic tree policies for reinforcement learning. *arXiv preprint arXiv:2405.14956*, 2024.

Lehman, J., Gordon, J., Jain, S., Ndousse, K., Yeh, C., and Stanley, K. O. Evolution through large models. In *Handbook of evolutionary machine learning*, pp. 331–366. Springer, 2023.

Lin, J., Zhu, C., Kneuertz, P. J., Bai, Y., and Xue, Y. Medcausalx: Adaptive causal reasoning with self-reflection for trustworthy medical vision-language models. *arXiv preprint arXiv:2603.23085*, 2026.

Lin, Y.-A., Lee, C.-T., Yang, C.-H., Liu, G.-T., and Sun, S.-H. Hierarchical programmatic option framework. *Advances in Neural Information Processing Systems*, 37: 126677–126724, 2024.

Liu, F., Tong, X., Yuan, M., Lin, X., Luo, F., Wang, Z., Lu, Z., and Zhang, Q. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *arXiv preprint arXiv:2401.02051*, 2024a.

Liu, M., Yu, C.-H., Lee, W.-H., Hung, C.-W., Chen, Y.-C., and Sun, S.-H. Synthesizing programmatic reinforcement learning policies with large language model guided search. *arXiv preprint arXiv:2405.16450*, 2024b.

Luo, L., Zhang, G., Xu, H., Yang, Y., Fang, C., and Li, Q. End-to-end neuro-symbolic reinforcement learning with textual explanations. *arXiv preprint arXiv:2403.12451*, 2024.

Ma, Y. J., Liang, W., Wang, G., Huang, D.-A., Bastani, O., Jayaraman, D., Zhu, Y., Fan, L., and Anandkumar, A. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.

Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

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. Mathematical discoveries from program search with large language models. *Nature*, 625 (7995):468–475, 2024.

Shindo, H., Delfosse, Q., Dhami, D. S., and Kersting, K. Blendrl: A framework for merging symbolic and neural policy learning. *arXiv preprint arXiv:2410.11689*, 2024.

Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Todd, G., Padula, A. G., Stephenson, M., Piette, É., Soemers, D. J., and Togelius, J. Gavel: Generating games via evolution and language models. *Advances in Neural Information Processing Systems*, 37:110723–110745, 2024.

Trivedi, D., Zhang, J., Sun, S.-H., and Lim, J. J. Learning to synthesize programs as interpretable and generalizable policies. *Advances in neural information processing systems*, 34:25146–25163, 2021.

Van Stein, N. and Bäck, T. Llamea: A large language model evolutionary algorithm for automatically generating metaheuristics. *IEEE Transactions on Evolutionary Computation*, 29(2):331–345, 2024.

Verma, A., Murali, V., Singh, R., Kohli, P., and Chaudhuri, S. Programmatically interpretable reinforcement learning. In *International conference on machine learning*, pp. 5045–5054. PMLR, 2018.

Verma, A., Le, H., Yue, Y., and Chaudhuri, S. Imitation-projected programmatic reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Vincze, M., Ferrarotti, L., Custode, L. L., Lepri, B., and Iacca, G. Smose: Sparse mixture of shallow experts for interpretable reinforcement learning in continuous control tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 20982–20990, 2025.

Wang, G., Xie, Y., Jiang, Y., Mandlekar, A., Xiao, C., Zhu, Y., Fan, L., and Anandkumar, A. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Wang, P., Hu, Y., Liu, X., Yang, J., Wang, H., and Wen, Z. Atlasva: Self-evolving visual skill memory for teacher-free vlm agents. *arXiv preprint arXiv:2605.17933*, 2026.

Weng, Y., Zhu, M., Xia, F., Li, B., He, S., Liu, S., Sun, B., Liu, K., and Zhao, J. Large language models are better reasoners with self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 2550–2575, 2023.

Xia, C. S., Deng, Y., Dunn, S., and Zhang, L. Demystifying llm-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE):801–824, 2025.

Xu, J., Fu, Y., Tan, S. H., and He, P. Aligning the objective of llm-based program repair. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pp. 2548–2560. IEEE, 2025.

Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., and Press, O. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

Zhang, K., Li, Z., Li, J., Li, G., and Jin, Z. Self-edit: Fault-aware code editor for code generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 769–787, 2023.

Zhang, Y., Ruan, H., Fan, Z., and Roychoudhury, A. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 1592–1604, 2024.

Zhong, L., Wang, Z., and Shang, J. Debug like a human: A large language model debugger via verifying runtime execution step by step. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 851–870, 2024.

Zhu, C., Lin, Y., Shao, J., Lin, J., and Wang, Y. Pathology-aware prototype evolution via llm-driven semantic disambiguation for multicenter diabetic retinopathy diagnosis. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 9196–9205, 2025.

Zhu, C., Lin, Y., Chen, S., Wang, Y., and Lin, J. Medeyes: Learning dynamic visual focus for medical progressive diagnosis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 13916–13924, 2026a.

Zhu, C., Zeng, J., Jiang, J., Lin, J., and Wang, Y. Medsynapse-v: Bridging visual perception and clinical intuition via latent memory evolution. *arXiv preprint arXiv:2604.26283*, 2026b.

A. Prompt Templates and Examples

To facilitate reproducibility, we provide the core prompt templates used by the Critic and Actor modules in the REFLEX framework. We use `tcolorbox` to present these prompts clearly.

Critic Prompt - Visual Diagnosis

System Role: You are an expert control systems engineer and visual diagnostician.
Input: 1. Task Description: {task_description} 2. Parent Policy Code: "'python {parent_code} "' 3. Fitness Score: {fitness_score} 4. Behavioral Evidence Image: [Attached Image]
Instruction: Analyze the provided Behavioral Evidence image. This image visualizes the execution of the parent policy. Your goal is to identify why the policy failed to achieve a perfect score.
Do NOT write any code. Output your analysis STRICTLY as a JSON object with the following keys: - `failure_mode`: A short (2-4 words) categorization of the error. - `visual_evidence`: What specific physical/geometric anomalies do you see in the image? - `root_cause`: What algorithmic flaw in the parent code likely caused this visual behavior? - `suggested_fix`: How should the code be modified to fix this?
Output Format: "'json { "failure_mode": "...", "visual_evidence": "...", "root_cause": "...", "suggested_fix": "..." } "'

Actor Prompt - Code Synthesis

System Role: You are an expert Python programmer specializing in control algorithms.
Input: 1. Task Description: {task_description} 2. Parent Policy Code: "'python {parent_code} "' 3. Critic's Diagnosis: "'json {critic_diagnosis} "' 4. Retrieved Skills (from Skill Memory): {retrieved_skills}
Instruction: Your task is to write a new, improved child policy. 1. Read the Critic's diagnosis carefully to understand the failure mode and the suggested fix. 2. Review the retrieved skills. If any skill snippet is relevant to the suggested fix (e.g., an energy-pumping heuristic), adapt and integrate it into your new code. 3. Write the complete, executable Python function for the new policy.
Output Format: Provide ONLY the Python code block. Do not include any conversational text.

```
def choose_action(state):
    # Your improved implementation here
    ...
```

B. Interpretability of Critic Diagnoses

A central claim of the REFLEX framework is that decoupling visual diagnosis from code repair yields an auditable and interpretable evolutionary trace. To substantiate this claim, we provide a detailed taxonomy of the Critic's JSON diagnoses, qualitative examples across different tasks, and an analysis of diagnostic robustness under ambiguous visual evidence.

B.1. Taxonomy of Diagnoses

The Critic is prompted to output a structured JSON object containing specific keys that enforce a rigorous diagnostic thought process. This taxonomy ensures that the Actor (and a human auditor) receives standardized, actionable feedback rather than unstructured text. The standard JSON schema includes:

- `failure_mode`: A high-level categorization of the error (e.g., "Late Braking", "Over-rotation", "High Sidelobes").
- `visual_evidence`: A description of the specific visual anomalies observed in the Behavioral Evidence (BE) image (e.g., "The phase portrait shows the pendulum trajectory spiraling outward but failing to close the loop at the upright position").
- `root_cause`: The hypothesized algorithmic flaw in the parent policy code that led to the observed visual evidence (e.g., "The proportional gain K_p is too low to overcome gravity near the top").
- `suggested_fix`: A natural language recommendation for the Actor on how to modify the code (e.g., "Increase K_p or introduce an energy-pumping heuristic when $|\theta| > \pi/2$").

B.2. Qualitative Examples

Table 5 presents representative examples of the Critic's diagnoses across the evaluated tasks. These examples demonstrate how the Critic grounds its algorithmic hypotheses in concrete visual observations.

Table 5. Representative examples of Critic JSON diagnoses across different tasks.

Task	Critic Diagnosis JSON (Abridged)

Lunar Lander	{
	"failure_mode": "Premature Engine Cutoff",
	"visual_evidence": "The lander's trajectory descends smoothly until $y \approx 0.2$, after which the main engine plume disappears and the lander crashes at high velocity.",
	"root_cause": "The altitude threshold for the braking heuristic is set too high, causing the policy to disable the main engine before touchdown.",
	"suggested_fix": "Lower the altitude threshold for main engine activation to $y < 0.1$ and ensure the descent rate condition allows for continuous braking."
	}

Pendulum	{
	"failure_mode": "Insufficient Energy Pumping",
	"visual_evidence": "The phase portrait shows the pendulum oscillating back and forth in the lower half-plane, never crossing the $\theta = \pm\pi/2$ horizontal axis.",
	"root_cause": "The policy is applying torque proportional to the angle, which fights the natural swing dynamics instead of amplifying them.",
	"suggested_fix": "Implement an energy-pumping strategy: apply torque in the direction of the current angular velocity when far from the upright position."
	}

CCAA Synthesis	{
	"failure_mode": "High Sidelobe Level (SLL)",
	"visual_evidence": "The radiation pattern cut shows a prominent first sidelobe at $\theta \approx 45^\circ$ that exceeds the -20 dB target, reaching approximately -15 dB.",
	"root_cause": "The amplitude taper drops off too abruptly near the outer rings, creating an aperture illumination that resembles a uniform distribution rather than a smooth taper.",
	"suggested_fix": "Smooth the amplitude taper by increasing the excitation of the penultimate ring and slightly decreasing the center element."
	}

B.3. Robustness and Auditability

A key advantage of the JSON diagnosis is that it allows human developers to immediately audit the evolutionary process. If a proposed child policy fails, the developer can inspect the JSON to determine whether the failure was due to a *perception error* (the Critic misread the BE) or a *synthesis error* (the Actor wrote buggy code based on a correct diagnosis).

In our empirical observations, the Critic's diagnostic quality remains robust for well-designed BE representations (like the Lunar Lander trajectories or CCAA radiation cuts). However, we note that diagnostic quality can degrade under ambiguous visual evidence. For example, in the Acrobot task, if the BE renders the double-pendulum links overlapping heavily during a chaotic swing, the Critic occasionally misidentifies the direction of the joint velocities, leading to an incorrect `root_cause` hypothesis. Because REFLEX decouples diagnosis from repair, these perception errors are explicitly logged in the `visual_evidence` field, making it trivial for researchers to identify the need for a clearer BE rendering function, rather than blindly tuning the LLM prompt.

C. Computational Resources and Execution Time

The REFLEX framework is a train-free evolutionary search method. Consequently, it does not require gradient backpropagation or traditional neural network training. The primary computational workload consists of two parts: (1) environment simulation and Behavioral Evidence (BE) rendering, and (2) inference calls to the Large Language Models (LLMs).

All experiments were conducted on a local workstation equipped with 8× NVIDIA GeForce RTX 3090 GPUs. However, because we utilized commercial API endpoints for the LLMs (`qwen3.6-plus` for the Actor and `qwen-vl-max` for the Critic), the local GPUs were primarily used for parallelizing the Gymnasium environment rollouts, BE image rendering, and CCAA array factor computations.

The wall-clock execution time of a single evolutionary campaign is heavily bottlenecked by API latency rather than local compute. On average, a single LLM call (Critic + Actor) takes approximately 3 to 5 seconds. For a standard control task budget of 100 LLM evaluations, a complete evolutionary run takes roughly 10 to 15 minutes. The lightweight nature of the local simulation allows multiple seeds to be evaluated concurrently across the available CPU/GPU resources without significant overhead.