REFLEX: Reflective Evolution from LLM Experience

Pan Wang
University of Science and Technology of China
diogenescask@gmail.com

Abstract

Large multimodal language models (MLLMs) have emerged as powerful tools for guiding evolutionary search toward interpretable programmatic policies. In existing program-evolution systems, however, reusable knowledge is usually carried by whole programs in the population, and it is difficult to trace how a visual observation led to a particular code change and its measured outcome. We present REFLEX, a train-free evolutionary framework that links these steps in one loop. A vision-enabled Critic turns task-specific behavioral evidence into a structured diagnosis; the diagnosis retrieves executable code snippets from a persistent Skill Memory; a text-only Actor writes the child program; and the child–parent fitness change updates the utility of each retrieved snippet. Every step is recorded in a single trace. Under matched backends and 100-call budgets over 10 paired seeds, REFLEX reaches the solve threshold in a median of 13, 20, and 24 LLM calls on Acrobot, Pendulum, and Lunar Lander, roughly half the calls required by official MLES and by a compute-matched Actor-only ablation. Frozen Skill Memory banks from Pendulum or Lunar Lander raise the final Acrobot score on all 10 paired seeds. On a 36-dimensional antenna-array design task with equal evaluation budgets and shared initialization, REFLEX reaches the harder 25.25 score threshold on 9/10 seeds, compared with at most 3/10 for CMA-ES, GA, and PSO. The source code is available at https://github.com/wangpan-ustc/REFLEX.

1 Introduction

Deep reinforcement learning (DRL) has achieved remarkable success in complex control tasks [20], yet its resulting policies are typically black-box neural networks that are notoriously difficult to interpret, verify, or manually patch [17, 28, 38, 45]. Unlike black-box approaches, programmatic policies express controllers as human-readable source code [36]. This paradigm retains the expressiveness of modern programming while ensuring the safety and auditability necessary for real-world applications [19, 34, 13, 56, 48, 46, 44, 47]. Recently, large language models (LLMs) have emerged as powerful evolutionary operators capable of discovering such policies [11, 41, 14, 23]. Multimodal Large Language Model-assisted Evolutionary Search (MLES) [10] 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 [40, 61].

However, in existing frameworks like MLES, each BE-conditioned mutation is still a single multimodal call that must simultaneously interpret complex visual trajectories, diagnose the underlying failure, and synthesize the corrective code [18, 22, 29, 54]. This *diagnosis–repair entanglement* creates an opaque feedback loop. When a proposed mutation fails, it is difficult to audit whether the model misinterpreted the visual evidence or simply generated flawed code [43, 52, 31, 33, 64]. Furthermore, reusable knowledge in these systems is mostly kept inside individual programs rather than as executable code whose usefulness is measured [32], so algorithmic insights and code heuristics discovered in previous trials are hard to carry into new campaigns [12, 53].

To address these fundamental bottlenecks, we introduce **REFLEX** (Reflective Evolution From LLM Experience), a fully train-free evolutionary framework that, following the reflection–generation

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

separation of prior systems such as Reflexion [29] and ReEvo [55], decouples visual diagnosis from code generation. In REFLEX, a specialized vision-enabled *Critic* first analyzes the Behavioral Evidence and emits a structured, traceable 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 and credited with the fitness change of the children that use them. 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) [5, 4]. Under matched 100-call budgets over 10 paired seeds, REFLEX reaches the solve threshold in a median of 13, 20, and 24 LLM calls on Acrobot, Pendulum, and Lunar Lander, compared with 41, 53, and 49 for official MLES and 28, 46, and 43 for a compute-matched Actor-only ablation. It also achieves a final Lunar Lander NWS of 1.082 ± 0.009, and in the antenna design task reaches a score of 25.25 on 9/10 seeds under equal evaluation budgets and shared initialization, compared with at most 3/10 for CMA-ES, GA, and PSO.

In summary, our main contributions are:

- We identify the diagnosis–repair entanglement bottleneck in multimodal LLM-guided evolution and propose REFLEX, which adopts a decoupled Critic-Actor design from prior reflective systems [29, 55] to make programmatic mutations traceable and targeted.
- We introduce a self-evolving Skill Memory mechanism that credits reusable code snippets with the fitness change they produce and enables cross-run programmatic knowledge transfer across evolutionary campaigns.
- We demonstrate that, under matched budgets, REFLEX achieves strong sample efficiency across discrete and continuous control environments and generalizes to a real-world engineering design task.

2 Related Work

LLM-Guided Evolutionary Search. LLMs are now widely used as mutation and crossover operators in evolutionary algorithms [15, 26, 18, 8, 30, 35]. FunSearch [26] evolves programs against an automated evaluator and keeps them in a program database; EoH [18] co-evolves natural-language thoughts and code; LLaMEA [35] generates and refines complete metaheuristics; ReEvo [55] adds short- and long-term verbal reflections that steer subsequent code generation; and AlphaEvolve [24] and its open-source counterpart OpenEvolve [27] combine several LLMs, evaluators, and a program database. MLES [10] brings visual Behavioral Evidence into this setting, with multiple BE-conditioned operators for programmatic policy search. In all of these systems the evolved object is a program and the main store of reusable knowledge is the population or program database, possibly accompanied by textual reflections. REFLEX shares this backbone and, like ReEvo and AlphaEvolve, uses distinct LLM roles, but also stores snippet-level executable code that is retrieved by the visual diagnosis and credited with the child–parent fitness change it produces.

Programmatic Policies and Agent Memory. Programmatic policies offer a transparent and verifiable alternative to black-box neural networks in deep reinforcement learning (DRL) [36, 37, 34]. Eureka [21] uses LLMs to write reward functions for a DRL inner loop, whereas REFLEX evolves the executable policy directly. Self-improving agents increasingly rely on persistent memory [62, 63]. Voyager [39] stores executable skills retrieved by natural-language description and verified by self-checking; Agentic Context Engineering [58] incrementally curates a textual playbook of strategies; ReasoningBank [25] distills reasoning memories from successful and failed trajectories; and EvoTest [9] evolves an entire agent configuration (prompt, memory, hyperparameters, and tool routines) between episodes. REFLEX stores executable snippets, retrieves them with BM25 queries formed from a visual diagnosis, updates their utility from measured child–parent fitness within a population-based search, and is evaluated for cross-run and cross-task transfer with frozen banks [51].

Self-Reflection and Visual Diagnostics. Self-reflection and critique improve LLM reasoning and coding [29, 2, 57, 7, 42, 50, 49, 16], and Trace [3] optimizes computational workflows from execution traces and rich feedback. Recent work has further examined the reliability of LLM-based evaluators, distinguishing systematic between-prompt variation from stochastic within-prompt sampling noise [6]. Most reflective frameworks consume text: execution logs, compiler errors, or traces [1, 59, 60]. In continuous control and physical design, scalar rewards and text logs often miss the geometric or spatial character of a failure. Like MLES, REFLEX reads rendered behavioral evidence; its vision-language Critic writes a structured diagnosis that is logged and reused as the retrieval query. We apply the same loop to classic control and to Concentric Circular Antenna Array (CCAA) synthesis [4, 5], an engineering design problem usually addressed with 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 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 decouples visual diagnosis from code generation, (2) task-specific Behavioral Evidence, and (3) a persistent Skill Memory. A UCB1 bandit selects among evolutionary operators. We write $\Pi_t \subset \mathcal{P}$ for the current finite population.

3.2 Critic-Actor Decoupling via Multi-turn Diagnostic Chain

When a single generation step both interprets the image and writes code, the record of a mutation does not show which of the two went wrong. Following prior reflective systems [29, 55], the Multi-turn Diagnostic Chain (MDC) assigns these steps to separate calls whose inputs and outputs are logged.

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. 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.

Each mutation records the diagnosed visual problem D_p, the retrieved skills S_p, and the child p' with its measured fitness, so a failed mutation can be traced to the diagnosis or to the code edit.

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

In population-based LLM-guided evolution, a useful heuristic usually persists only inside the programs that contain it. REFLEX additionally stores such heuristics as executable snippets with a fitness-grounded utility, and uses an operator bandit to allocate mutations.

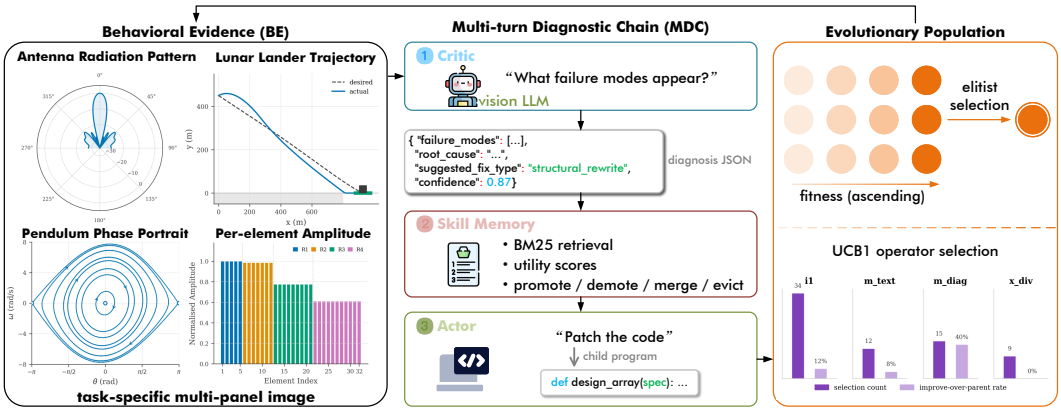


Figure 1: Overview of the REFLEX framework. The evolutionary loop is driven by a Multi-turn Diagnostic Chain (MDC) that decouples visual diagnosis from code generation. First, task-specific Behavioral Evidence (BE) is analyzed by a vision-enabled Critic to produce a structured JSON diagnosis. Next, a 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.

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:

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

where $f(q, k_i)$ is the term frequency of query term q in trigger k_i. Retrieval ranks skills by this score alone. After a child p' of parent p is evaluated, each retrieved skill that the Actor reports using is credited with $\Delta = f_{p'} - f_p$ ($\Delta = -1$ for an invalid child): $u_i \leftarrow u_i + \min(\Delta, 1)$ if $\Delta > 0$, and $u_i \leftarrow u_i - \frac{1}{2}\min(|\Delta|, 1)$ otherwise. New skills start at $u_i = 1$. A skill is archived when $u_i < -1$ after at least three retrievals, near-duplicate skills are merged by one LLM call, and when the bank exceeds its capacity of 256 active skills the lowest-utility skills are archived. Besides these archive rules, u_i is shown to the Actor next to each retrieved snippet, selects the skills shown to `i1`, and selects the frozen banks used for transfer.

New skills are extracted only when a child improves on its parent ($\Delta > 0$). In such cases, one LLM call abstracts the child's code, its concept, and the diagnosis it addressed into a short snippet c_i and a one-sentence trigger k_i.

To manage the evolutionary loop, REFLEX uses five operators, which form $\mathcal{O}$ in this order: diagnostic mutation through the MDC (`m_diag`); text-only mutation from the parent's code and score (`m_text`); crossover of three mutually diverse population members (`x_div`); generation of a new program from scratch (`i1`); and cross-domain inspiration (`m_inspire`), which asks the Actor to transfer a mechanism from a randomly drawn unrelated domain into the parent. We model operator selection as a multi-armed bandit solved with UCB1. At step t, the framework selects $o_t \in \arg\max_{o \in \mathcal{O}} U_t(o)$, where

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

$\hat{\mu}_o$ is the mean normalized reward of operator o, N_o is the number of times o has been selected, and C balances exploration and exploitation. Untried operators ($N_o = 0$) are selected first, and ties are broken by the order of $\mathcal{O}$ listed above. To satisfy the bounded-reward assumption of UCB1, the fitness delta Δ is clipped to $[0, \Delta_{\max}]$ and divided by $\Delta_{\max}$, the largest positive delta observed so far in the run. In the control tasks, a positive reward r is also credited, as $0.5^d r$, to the operators that produced the child's ancestors at depths $d \leq 3$, and after 12 consecutive children without a new best, REFLEX runs a forced burst of one `m_inspire` and one `i1` proposal.

As illustrated in Figure 2, the UCB1 bandit dynamically adapts to the search phase (Figure 2a), while the Skill Memory continuously evaluates extracted code snippets (Figure 2b). Frozen snapshots of

Algorithm 1 REFLEX: Reflective Evolution from LLM Experience (as run in the control tasks)

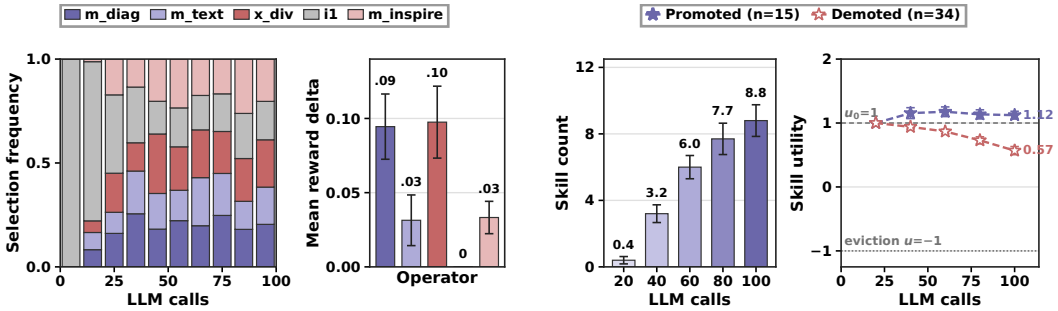
Input: task $\mathcal{T}$, fitness $\mathcal{F}$, BE renderer $\mathcal{B}$, template program p_0, population capacity N, operators $\mathcal{O}$, stagnation threshold $s_{\max}$, LLM-call budget B (every Critic, Actor, repair, revision, abstraction, merge, and failed call counts)
Evaluate p_0 and cache (f_{p_0}, V_{p_0}); $\Pi_t \leftarrow \{p_0\}$; $\mathcal{M} \leftarrow \mathcal{M}_0$ (empty or a frozen bank); $N_o, \hat{\mu}_o \leftarrow 0\ \forall o \in \mathcal{O}$; $s \leftarrow 0$
Fill Π_t to $N/2$ members with diagnostic or text-only mutations of p_0
while fewer than B LLM calls have been attempted **do**
  if $s \geq s_{\max}$ **then**
    $Q \leftarrow (\texttt{m_inspire}, \texttt{i1})$; $s \leftarrow 0$ {forced burst}
  else
    $Q \leftarrow (o_t)$ with $o_t \in \arg\max_{o \in \mathcal{O}} U_t(o)$ (Eq. 2)
  end if
  for each operator o in Q **do**
    $p \leftarrow \textsc{Tournament}_3(\Pi_t)$; $D_p \leftarrow \varnothing$; $S_p \leftarrow \emptyset$
    if $o = \texttt{m_diag}$ **then**
      $D_p \sim P_{\text{critic}}(\cdot \mid V_p, f_p, \mathcal{T})$ {reused for at most two Actor calls on p}
      $S_p \leftarrow$ top-K BM25 skills for D_p in $\mathcal{M}$; $p' \sim P_{\text{actor}}(\cdot \mid p, D_p, S_p, \mathcal{T})$
    else if $o = \texttt{x_div}$ **then**
      $p' \sim P_{\text{actor}}(\cdot \mid$ three mutually diverse members of $\Pi_t, \mathcal{T})$
    else
      $p' \sim P_{\text{actor}}(\cdot \mid p, f_p, \mathcal{T})$ for `m_text`; the same plus a foreign-domain mechanism for `m_inspire`; a new program shown the top-3 skills by u_i for `i1`
    end if
    Evaluate p' and cache $(f_{p'}, V_{p'})$; if it crashes, make one repair call and re-evaluate
    If p is near the best and p' regressed, one revision call yields a second child, processed like p' below
    $\Delta \leftarrow f_{p'} - f_p$, or $\Delta \leftarrow -1$ if p' is invalid
    Update u_i of the skills in S_p that the Actor reports using, then apply the archive rules (Sec. 3.4)
    if $\Delta > 0$ **then** abstract one new skill from p' (one LLM call) and add it to $\mathcal{M}$
    Update $N_o, \hat{\mu}_o$ with $r = \min(\max(\Delta, 0), \Delta_{\max})/\Delta_{\max}$; if $r > 0$, credit ancestors' operators with $0.5^d r$
    Insert p' into Π_t if valid and not a duplicate, then keep the N best
    $s \leftarrow 0$ if $f_{p'}$ is a new best, else $s \leftarrow s + 1$
  end for
end while
Return: $p^* = \arg\max_{p \in \Pi_t} f_p$

the Skill Memory can be loaded into new campaigns, which we use for the controlled transfer study in Section 4.4. 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 tasks. Lunar Lander uses a Normalized Weighted Score (NWS) combining reward, fuel use, and success rate; Pendulum uses a stable-upright score; and Acrobot reports the raw evaluator score (higher is better, target -0.20). The fourth task is Concentric Circular Antenna Array (CCAA) synthesis, 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. For the control tasks we report the final best score, the AUC of the best-so-far curve over the call budget, the number of seeds reaching the task target (Solved), and the number of LLM calls needed to first reach it (calls to solve).



(a) Operator selection frequency and reward delta.

(b) Skill memory size and utility tracking.

Figure 2: Evolutionary dynamics in REFLEX. (a) The UCB1 bandit dynamically balances diagnostic mutation (`m_diag`), text mutation (`m_text`), crossover (`x_div`), cross-domain inspiration (`m_inspire`), and initialization (`i1`) based on their historical success. (b) The Skill Memory continuously evaluates extracted code snippets, promoting skills whose children improve on their parents, demoting the others, and archiving skills whose utility falls below -1.

Baselines and Protocol. We compare REFLEX with two baselines under matched conditions: (i) official MLES [10], rerun from its public implementation with its native prompts, operators, parent and environmental selection, and duplicate handling; and (ii) Actor-only, a compute-matched ablation that removes the Critic and spends the saved calls on additional Actor proposals. Its `m_diag` operator keeps its bandit arm, but the Actor receives a placeholder diagnosis that lists only the parent's scalar evaluation metrics, and this text also serves as the BM25 query; no BE image reaches any LLM call, and all other operators and Skill Memory rules are unchanged. All methods use the same model weights (Qwen3-VL-235B-A22B-Instruct) for every LLM role, population size 16, a 4,096-token output cap, identical evaluators. The budget counts every LLM generation attempt, including Critic, Actor, repair, abstraction, and failed calls. The 50- and 100-call budgets are separate runs with different random seeds; within each budget, comparisons are paired by seed: we report 95% paired-bootstrap confidence intervals, exact Wilcoxon signed-rank tests with Holm correction within each planned family, and Cohen's d_z. The cross-task transfer study (Section 4.4) uses Qwen3.5-27B. All hyperparameters and protocol details are listed in Appendix C.

4.2 Sample-Efficient Policy Discovery

Table 1 reports the matched comparison, and Figure 3(a) summarizes the median calls to solve at the 100-call budget. At 50 calls, REFLEX has the highest final score, AUC, and solve count on all three tasks. On Lunar Lander and Pendulum, its paired AUC gains over both baselines are significant at both budgets (Holm-corrected $p \leq 0.032$); for example, on Pendulum at 50 calls the AUC gain is $+0.125$ [0.045, 0.205] over Actor-only ($d_z = 0.90$) and $+0.160$ [0.070, 0.250] over MLES ($d_z = 1.10$). Final-score gains have 95% CIs excluding zero on Pendulum at both budgets and on Lunar Lander at 50 calls (e.g., $+0.037$ [0.008, 0.066] over Actor-only), but not on Lunar Lander at 100 calls, where all three methods reach a similar final NWS. On Acrobot (without paired tests), REFLEX solves every seed at both budgets with a median of 12–13 calls, versus 24–28 for Actor-only and 35–41 for MLES.

REFLEX spends about 40% of its calls on the Critic and therefore evaluates fewer policies: 27 on average in a 50-call Lunar Lander campaign, versus 48 for Actor-only and 42 for MLES. It also produces fewer invalid programs; on Acrobot at 50 calls, 2 of 360 REFLEX children (0.6%) are invalid, versus 75 of 510 (14.7%) for MLES. With a 20-call cap on Acrobot, REFLEX and Actor-only each solve 6/10 seeds and their paired differences are not significant (Actor-only minus REFLEX: final $+0.015$, 95% CI $[-0.123, 0.453]$, $p = 0.92$; AUC $+0.020$, $p = 0.63$); the Critic calls pay off once the budget allows several diagnosis-conditioned mutations.

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

Table 1: Matched comparison under 50 and 100 total LLM calls (same model weights for all roles, population 16, 10 paired seeds; every LLM attempt counts). Final and AUC are mean ± SEM, except in the Acrobot 50-call block, where Final is the median (†) and AUC is the mean without SEM. Solved counts seeds that reach the task target; Calls is the median [IQR] number of LLM calls needed to first reach it. Lunar Lander reports NWS, Pendulum the stable-upright score, and Acrobot the raw evaluator score (target −0.20). Best value in each budget block is in **bold**.

		Lunar Lander (NWS)				Pendulum				Acrobot (raw)			
Budget	**Method**	Final↑	AUC↑	Solved	Calls↓	Final↑	AUC↑	Solved	Calls↓	Final↑	AUC↑	Solved	Calls↓
50	Official MLES	$1.010_{\pm 0.020}$	$0.860_{\pm 0.028}$	6/10	41 [32,47]	$0.735_{\pm 0.050}$	$0.575_{\pm 0.046}$	6/10	39 [29,48]	$-0.145^{\dagger}$	-0.500	9/10	35 [23,47]
	Actor-only	$1.018_{\pm 0.018}$	$0.895_{\pm 0.024}$	7/10	35 [27,43]	$0.770_{\pm 0.045}$	$0.610_{\pm 0.042}$	6/10	34 [25,45]	$-0.148^{\dagger}$	-0.460	9/10	24 [15,36]
	REFLEX	$\mathbf{1.055}_{\pm 0.012}$	$\mathbf{0.965}_{\pm 0.018}$	**9/10**	**22** [16,29]	$\mathbf{0.875}_{\pm 0.030}$	$\mathbf{0.735}_{\pm 0.035}$	**9/10**	**18** [12,27]	$\mathbf{-0.135}^{\dagger}$	$\mathbf{-0.320}$	**10/10**	**12** [7,20]
100	Official MLES	$1.069_{\pm 0.011}$	$0.955_{\pm 0.018}$	9/10	49 [36,66]	$0.860_{\pm 0.034}$	$0.715_{\pm 0.036}$	8/10	53 [38,72]	$-0.165_{\pm 0.015}$	$-0.490_{\pm 0.055}$	9/10	41 [27,61]
	Actor-only	$1.064_{\pm 0.012}$	$0.970_{\pm 0.017}$	9/10	43 [32,58]	$0.875_{\pm 0.030}$	$0.750_{\pm 0.032}$	9/10	46 [31,63]	$-0.155_{\pm 0.012}$	$-0.410_{\pm 0.050}$	**10/10**	28 [17,43]
	REFLEX	$\mathbf{1.082}_{\pm 0.009}$	$\mathbf{1.025}_{\pm 0.012}$	**10/10**	**24** [17,31]	$\mathbf{0.935}_{\pm 0.020}$	$\mathbf{0.845}_{\pm 0.025}$	**10/10**	**20** [13,29]	$\mathbf{-0.145}_{\pm 0.008}$	$\mathbf{-0.280}_{\pm 0.035}$	**10/10**	**13** [7,21]

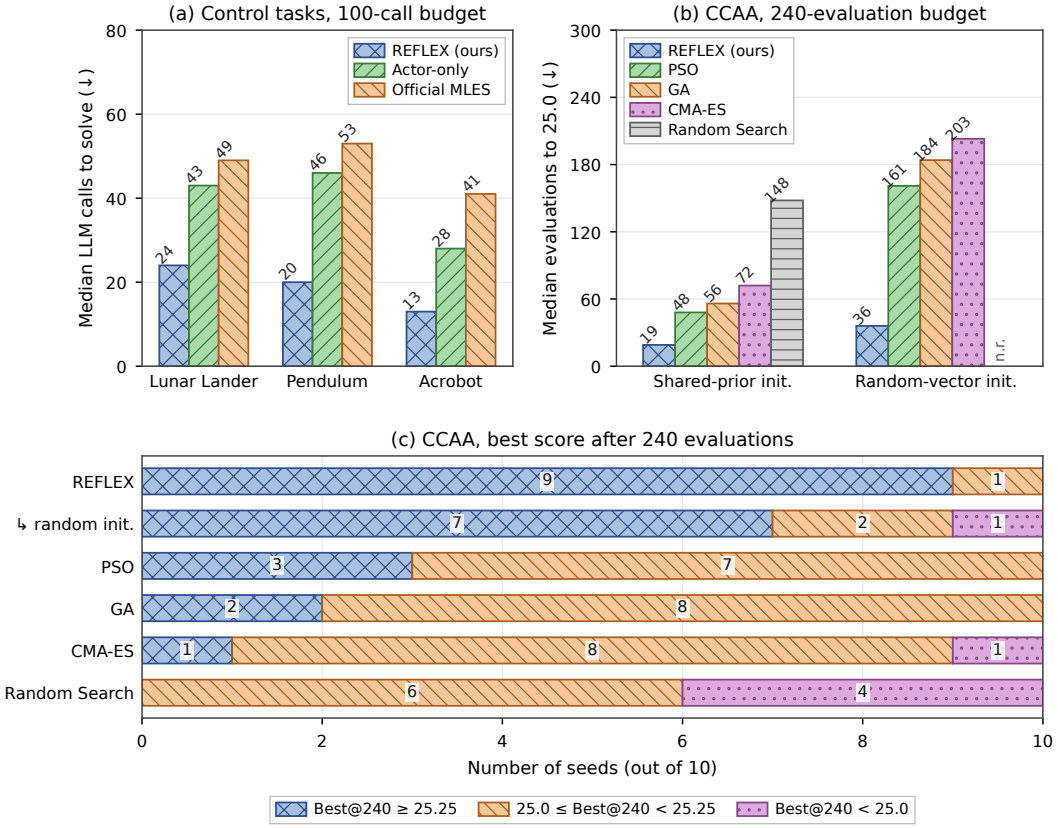


Figure 3: Sample efficiency under matched budgets (10 paired seeds; lower is better in (a) and (b)). **(a)** Median LLM calls to solve on the control tasks with the same backend and a cap of 100 total LLM calls, where every Critic, Actor, repair, and failed attempt counts. **(b)** Median objective evaluations to reach a CCAA score of 25.0 under an equal 240-evaluation budget, with shared-prior or random-vector initialization; n.r. marks a method that never reaches 25.0. **(c)** Number of CCAA seeds whose Best@240 falls in each score band. All rows use shared-prior initialization except "↪ random init.", which denotes REFLEX with random-vector initialization.

strong starting points for low-sidelobe arrays [5], and modern CCAA synthesis studies optimize radii, rotations, and excitations to further reduce sidelobes [4].

REFLEX uses the 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.

Because the LLM already knows common array designs, we give REFLEX, Random Search, CMA-ES, GA, and PSO the same initialization on the same evaluator: either random design vectors, or a shared prior consisting of the same first LLM/analytical designs for every method. All methods receive the same budget of 240 objective evaluations, and every evaluation counts, including REFLEX's inner local-search evaluations. Table 2 and Figure 3(b,c) report 10 paired seeds. The shared prior helps every classical optimizer substantially (e.g., GA reaches 25.0 on 10/10 seeds instead of 4/10). Under both initializations REFLEX is the most sample-efficient method: with the shared prior it reaches 25.0 in a median of 19 evaluations, versus 48–148 for the baselines, and attains a Best@240 of 25.33 ± 0.02. The gap is largest at the harder 25.25 threshold, which REFLEX reaches on 9/10 seeds with the shared prior and 7/10 from random vectors, while no classical optimizer exceeds 3/10 in either setting.

Table 2: CCAA synthesis under an equal budget of 240 objective evaluations (10 paired seeds; all evaluations count, including REFLEX's inner local search). Best@k is the best score after k evaluations (mean $\pm$ SEM). The threshold columns count seeds whose Best@240 reaches 25.0 or 25.25; Evals-to-25.0 is the median [IQR] over seeds that reach it (–: never reached).

Method	Initialization	Best@50↑	Best@100↑	Best@240↑	≥ 25.0	≥ 25.25	Evals-to-25.0↓
Random Search	Random vector	23.82 ± 0.18	24.10 ± 0.15	24.48 ± 0.12	0/10	0/10	–
CMA-ES	Random vector	24.28 ± 0.13	24.55 ± 0.11	24.83 ± 0.09	2/10	0/10	203 [175, 231]
GA	Random vector	24.34 ± 0.12	24.66 ± 0.10	24.95 ± 0.07	4/10	0/10	184 [136, 220]
PSO	Random vector	24.40 ± 0.11	24.72 ± 0.09	25.00 ± 0.06	5/10	0/10	161 [119, 206]
REFLEX	Random vector	$\mathbf{24.86 \pm 0.06}$	$\mathbf{25.11 \pm 0.04}$	$\mathbf{25.28 \pm 0.03}$	**9/10**	**7/10**	**36** [22, 57]
Random Search	Shared prior	24.86 ± 0.07	24.94 ± 0.06	25.03 ± 0.05	6/10	0/10	148 [104, 211]
CMA-ES	Shared prior	24.94 ± 0.05	25.04 ± 0.04	25.12 ± 0.03	9/10	1/10	72 [48, 119]
GA	Shared prior	24.96 ± 0.04	25.07 ± 0.03	25.18 ± 0.03	**10/10**	2/10	56 [37, 91]
PSO	Shared prior	24.98 ± 0.04	25.10 ± 0.03	25.21 ± 0.02	**10/10**	3/10	48 [31, 78]
REFLEX	Shared prior	$\mathbf{25.02 \pm 0.04}$	$\mathbf{25.18 \pm 0.03}$	$\mathbf{25.33 \pm 0.02}$	**10/10**	**9/10**	**19** [11, 31]

Figure 5 provides a detailed analysis of the optimized array. The elevation and azimuthal cuts (Figure 5a) demonstrate that the REFLEX-best design achieves lower sidelobe levels compared to classical Dolph-Chebyshev and Taylor tapers. Furthermore, the Pareto front (Figure 5b) illustrates the trade-off between sidelobe level and directivity explored during the evolutionary search, highlighting the target design band. Figure 4 visualizes the resulting 3D radiation pattern.

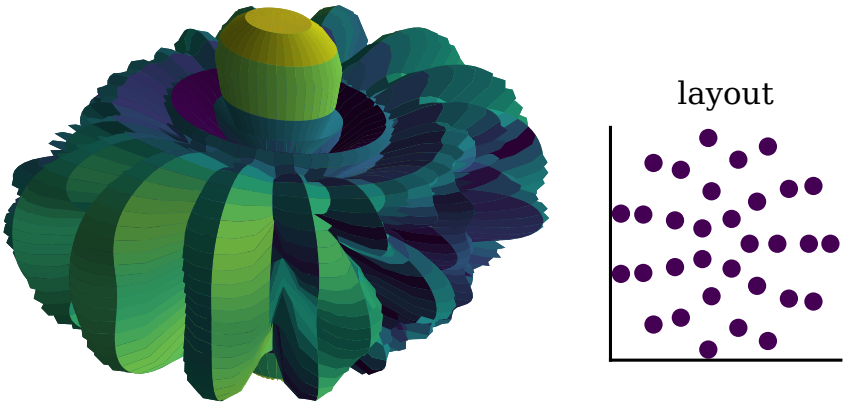
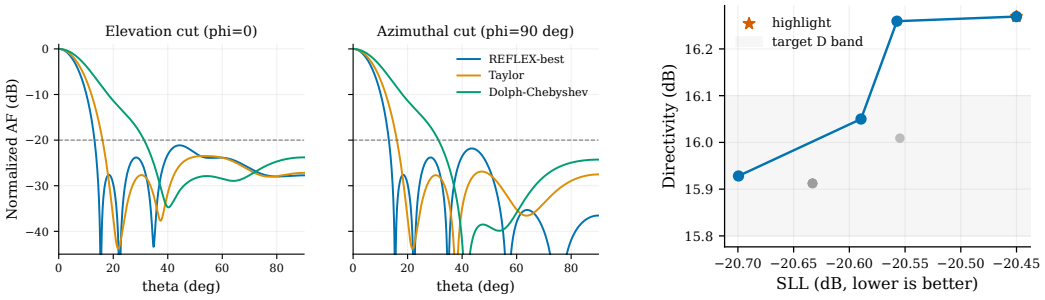


Figure 4: Best compact CCAA radiation pattern discovered in the existing REFLEX campaign.

4.4 Controlled Skill Memory Transfer

We test whether executable Skill Memory transfers across tasks with a controlled three-condition study on Acrobot: an empty bank (*cold*), the top-2 skills from Pendulum campaigns, and the top-2 skills from Lunar Lander campaigns. Both source banks are frozen and hash-verified before the target



(a) Elevation and azimuthal cuts.

(b) SLL vs. Directivity Pareto front.

Figure 5: Detailed analysis of the CCAA synthesis results. (a) Radiation pattern cuts comparing the REFLEX-best design against classical Dolph-Chebyshev and Taylor tapers, showing superior sidelobe suppression. (b) The Pareto front of discovered designs, illustrating the trade-off between Sidelobe Level (SLL) and Directivity within the target band.

runs, selected by the same source-task utility rule, and audited to exclude Acrobot-specific identifiers. Each condition uses the same model (Qwen3.5-27B), a 20-call budget, and 10 paired campaign seeds; all 30 runs and 600 model attempts passed validation.

In Figure 7, boxes show medians and IQRs, whiskers span min–max, and linked dots indicate paired seeds across conditions. Both source banks raise the final score on every seed (10/10 paired wins each; exact Wilcoxon $p = 0.002$), from a cold mean of -0.139 to -0.105 (Pendulum) and -0.118 (Lunar Lander). AUC@20 improves on 9/10 seeds for each bank, with means increasing from -0.523 to -0.310 and -0.380, respectively, although the paired test is significant only for the Pendulum bank ($p = 0.049$; Lunar Lander $p = 0.065$). Median calls-to-target fall from 4.0 to 2.0 and 2.5, but this speedup is consistent only for the Pendulum bank (6/2/2 W/T/L, $p = 0.039$); with the Lunar Lander bank two seeds need 16–17 calls ($p = 0.91$). The trace shows that transfer works through the Actor rewriting retrieved code rather than copying it: Pendulum skills were used in 58 evaluated Acrobot children (13 improvements, 45 regressions) and Lunar Lander skills in 14 (5 improvements, 9 regressions). Fitness selection keeps the few useful adaptations, so transfer improves whole campaigns even though most individual reuses cause regressions.

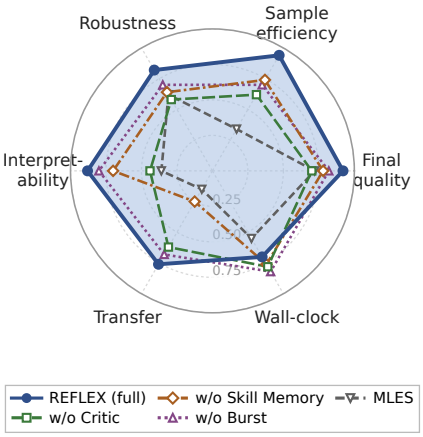


Figure 6: Qualitative component profile across sample efficiency, quality, robustness, interpretability, transfer, and wall-clock time axes.

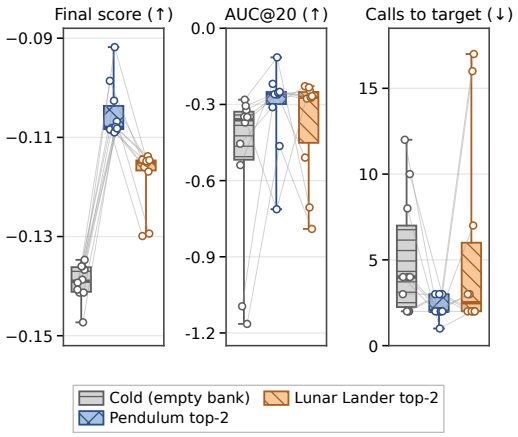


Figure 7: Cross-task Skill Memory transfer from Pendulum and Lunar Lander to Acrobot (10 paired seeds, 20-call budget).

4.5 Ablation Studies

The compute-matched Actor-only ablation on all three control tasks is part of Table 1. Table 3 adds two Acrobot ablations with matched call accounting and 10 paired seeds.

Table 3: Acrobot ablations (10 paired seeds; Target counts seeds reaching -0.20). **(a)** Behavioral Evidence given to the Critic, under an identical call budget for all conditions; values are mean $\pm$ SEM. **(b)** Object stored in a frozen source Skill Memory, with a 20-call cap; Final is the median and AUC@20 the mean over seeds.

(a) Behavioral Evidence perturbation

Condition	**Final↑**	**AUC↑**	**Target**	**Calls-to-target↓**
Full REFLEX (correct BE)	$\mathbf{-0.1352 \pm 0.0041}$	$\mathbf{-0.2850 \pm 0.0337}$	10/10	$\mathbf{5.50 \pm 1.58}$
Shuffled-BE Critic	-0.1426 ± 0.0040	-0.4270 ± 0.0556	10/10	14.70 ± 4.37
Blank-BE Critic	-0.1388 ± 0.0020	-0.3620 ± 0.0408	10/10	9.50 ± 2.55
Actor-only (no Critic)	-0.1373 ± 0.0021	-0.3450 ± 0.0347	10/10	10.30 ± 3.34

(b) Memory object

Memory	**Final↑**	**AUC@20↑**	**Target**	**Invalid children↓**
Executable fragments	$\mathbf{-0.1363}$	-0.5445	9/10	**0/163**
Textual strategies	-0.1393	$\mathbf{-0.4742}$	**10/10**	1/166
No memory	-0.1417	-0.5618	9/10	3/160

Behavioral Evidence. To separate the effect of the visual evidence from that of an extra Critic call, we keep the Critic but give it a shuffled BE image that does not belong to the parent, or a blank image. Correct BE gives the best AUC and the fastest target attainment (Table 3a). Its paired AUC gain is $+0.142$ [0.095, 0.181] over shuffled BE ($p = 0.004$), $+0.077$ [0.040, 0.110] over blank BE ($p = 0.012$), and $+0.060$ [0.027, 0.089] over Actor-only ($p = 0.012$), and it needs 9.2, 4.0, and 4.8 fewer calls to reach the target, respectively. The final-score gain over Actor-only is not significant ($+0.0021$ [-0.0005, 0.0046], $p = 0.18$).

Memory object. With the same frozen source bank and a 20-call cap, executable fragments, textual strategies, and no memory lead to different search profiles (Table 3b), but no pairwise difference is significant (textual minus executable, median of the paired differences: final -0.0010 [-0.0093, 0.0087], $p = 0.82$; AUC $+0.017$ [-0.194, 0.265], $p = 0.70$). Executable fragments produce no invalid children (0/163).

Figure 6 provides a qualitative component profile of the Lunar Lander variants, in which the full REFLEX framework scores highest on sample efficiency, final quality, robustness, and interpretability.

5 Conclusion and Limitations

REFLEX links visual Behavioral Evidence, a structured diagnosis, retrieved executable skills, the generated child, and its measured fitness in one loop, and credits each retrieved snippet with the fitness change it produces. With matched backends and total-call budgets of 50–100 calls, it reaches solve thresholds in roughly half the calls of official MLES and of a compute-matched Actor-only ablation. Shuffling or blanking the Behavioral Evidence removes this gain, and frozen Skill Memory banks from two source tasks improve search on Acrobot. The same loop also applies to a 36-dimensional antenna design task.

Limitations. At a 20-call cap on Acrobot, the Critic brings no measurable benefit over Actor-only. Our control tasks are low-dimensional and CCAA is a single structured design problem; building informative BE for high-dimensional observations or long-horizon control remains open. We have not run a human study of whether individual diagnoses are correct, nor a sensitivity study of population size, the UCB1 constant, or memory capacity. Cross-task transfer is tested on two source–target pairs with one model, and executable memory does not outperform textual memory in our ablation. REFLEX also inherits the stochasticity and cost of LLM inference.

Acknowledgments and Disclosures

This work received no third-party funding or support. The authors declare no competing interests.

References

[1] Islem Bouzenia, Premkumar Devanbu, and Michael Pradel. Repairagent: An autonomous, llm-based agent for program repair. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pages 2188–2200. IEEE, 2025.

[2] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. *arXiv preprint arXiv:2304.05128*, 2023.

[3] Ching-An Cheng, Allen Nie, and Adith Swaminathan. Trace is the next autodiff: Generative optimization with rich feedback, execution traces, and llms. *Advances in Neural Information Processing Systems*, 37:71596–71642, 2024.

[4] Nihad Dib and Ashraf Sharaqa. 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.

[5] Charles L Dolph. 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.

[6] Pengbin Feng, Chunlei Meng, Daozheng Qu, Zhilin Zhang, Haoran Liu, and Jiekai Wu. Second-order response laws for llm judges: Debiased estimation of prompt instability. *arXiv preprint arXiv:2608.16253*, 2026.

[7] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023.

[8] Qingyan Guo, Rui Wang, Junliang Guo, Bei Li, Kaitao Song, Xu Tan, Guoqing Liu, Jiang Bian, and Yujiu Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. *arXiv preprint arXiv:2309.08532*, 2023.

[9] Yufei He, Juncheng Liu, Yue Liu, Yibo Li, Tri Cao, Zhiyuan Hu, Xinxing Xu, and Bryan Hooi. Evotest: Evolutionary test-time learning for self-improving agentic systems. In *International Conference on Learning Representations*, volume 2026, pages 104166–104202, 2026.

[10] Qinglong Hu, Xialiang Tong, Mingxuan Yuan, Fei Liu, Zhichao Lu, and Qingfu Zhang. Multimodal llm-assisted evolutionary search for programmatic control policies. *arXiv preprint arXiv:2508.05433*, 2025.

[11] Yihao Hu, Zhihao Wen, Xiujin Liu, Pan Wang, Xin Zhang, and Wei Wu. Seal: Synergistic co-evolution of agents and learning environments. *arXiv preprint arXiv:2605.24426*, 2026.

[12] Nan Jiang, Xiaopeng Li, Shiqi Wang, Qiang Zhou, Soneya B Hossain, Baishakhi Ray, Varun Kumar, Xiaofei Ma, and Anoop Deoras. Ledex: Training llms to better self-debug and explain code. *Advances in Neural Information Processing Systems*, 37:35517–35543, 2024.

[13] Hector Kohler, Quentin Delfosse, Riad Akrour, Kristian Kersting, and Philippe Preux. Interpretable and editable programmatic tree policies for reinforcement learning. *arXiv preprint arXiv:2405.14956*, 2024.

[14] Zhiyi Kuang, Ryan Rong, YuCheng Yuan, and Allen Nie. Learning game-playing agents with generative code optimization. *arXiv preprint arXiv:2508.19506*, 2025.

[15] Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. Evolution through large models. In *Handbook of evolutionary machine learning*, pages 331–366. Springer, 2023.

[16] Jianxin Lin, Chunzheng Zhu, Peter J Kneuertz, Yunfei Bai, and Yuan Xue. When models learn to ask why: Adaptive causal reasoning for trustworthy medical vision-language models. *arXiv preprint arXiv:2603.23085*, 2026.

[17] Yu-An Lin, Chen-Tao Lee, Chih-Han Yang, Guan-Ting Liu, and Shao-Hua Sun. Hierarchical programmatic option framework. *Advances in Neural Information Processing Systems*, 37: 126677–126724, 2024.

[18] Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. *arXiv preprint arXiv:2401.02051*, 2024.

[19] Max Liu, Chan-Hung Yu, Wei-Hsu Lee, Cheng-Wei Hung, Yen-Chun Chen, and Shao-Hua Sun. Synthesizing programmatic reinforcement learning policies with large language model guided search. *arXiv preprint arXiv:2405.16450*, 2024.

[20] Lirui Luo, Guoxi Zhang, Hongming Xu, Yaodong Yang, Cong Fang, and Qing Li. End-to-end neuro-symbolic reinforcement learning with textual explanations. *arXiv preprint arXiv:2403.12451*, 2024.

[21] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931*, 2023.

[22] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

[23] Allen Nie, Xavier Daull, Zhiyi Kuang, Abhinav Akkiraju, Anish Chaudhuri, Max Piasevoli, Ryan Rong, YuCheng Yuan, Prerit Choudhary, Shannon Xiao, et al. Understanding the challenges in iterative generative optimization with llms. *arXiv preprint arXiv:2603.23994*, 2026.

[24] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco JR Ruiz, Abbas Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

[25] Siru Ouyang, Jun Yan, I Hsu, Yanfei Chen, Ke Jiang, Zifeng Wang, Rujun Han, Long Le, Samira Daruki, Xiangru Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pages 94327–94354, 2026.

[26] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

[27] Asankhaya Sharma. Openevolve: an open-source evolutionary coding agent. *GitHub repository. Accessed*, pages 08–10, 2025.

[28] Hikaru Shindo, Quentin Delfosse, Devendra Singh Dhami, and Kristian Kersting. Blendrl: A framework for merging symbolic and neural policy learning. *arXiv preprint arXiv:2410.11689*, 2024.

[29] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

[30] Graham Todd, Alexander G Padula, Matthew Stephenson, Éric Piette, Dennis J Soemers, and Julian Togelius. Gavel: Generating games via evolution and language models. *Advances in Neural Information Processing Systems*, 37:110723–110745, 2024.

[31] Yongqi Tong, Dawei Li, Sizhe Wang, Yujia Wang, Fei Teng, and Jingbo Shang. Can llms learn from previous mistakes? investigating llms' errors to boost for reasoning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3065–3080, 2024.

[32] Yongqi Tong, Pan Wang, Hang Wang, Jianshe Li, Xin Zhang, Jiang-Ming Yang, and Wei Wu. Grounded skill synthesis from code at scale for agentic intelligence. *arXiv preprint arXiv:2609.05571*, 2026.

[33] Yongqi Tong, Zhenyu Zhang, Zimi Liu, Kewei Fu, Mingli Song, Haofei Zhang, Junshao Zhang, Hong Zhu, Jiang-Ming Yang, Xin Zhang, et al. Ask, condition or abstain: Reinforcement learning for missing-premise reasoning. *arXiv preprint arXiv:2608.16554*, 2026.

[34] Dweep Trivedi, Jesse Zhang, Shao-Hua Sun, and Joseph J Lim. Learning to synthesize programs as interpretable and generalizable policies. *Advances in neural information processing systems*, 34:25146–25163, 2021.

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

[36] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International conference on machine learning*, pages 5045–5054. PMLR, 2018.

[37] Abhinav Verma, Hoang Le, Yisong Yue, and Swarat Chaudhuri. Imitation-projected programmatic reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

[38] Mátyás Vincze, Laura Ferrarotti, Leonardo Lucio Custode, Bruno Lepri, and Giovanni Iacca. 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, pages 20982–20990, 2025.

[39] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[40] Pan Wang, Yihao Hu, Xiujin Liu, Jingchu Yang, Hang Wang, and Zhihao Wen. Atlasva: Self-evolving visual skill memory for teacher-free vlm agents. *arXiv preprint arXiv:2605.17933*, 2026.

[41] Pan Wang, Yihao Hu, Hang Wang, Zirui Lv, Xin Zhang, Jianshe Li, Jiang-Ming Yang, Wei Wu, and Yongqi Tong. Diagnosis before recovery: Turning agent failures into selective self-correction. *arXiv preprint arXiv:2608.11772*, 2026.

[42] Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 2550–2575, 2023.

[43] Chunqiu Steven Xia, Yinlin Deng, Soren Dunn, and Lingming Zhang. Demystifying llm-based software engineering agents. *Proceedings of the ACM on Software Engineering*, 2(FSE): 801–824, 2025.

[44] Linqi Xiao, Abhishek Bhattacharyya, Aparaajitha Gomathinayakam Latha, Venkateswarlu Gudepu, and Andrea Fumagalli. Robotic process automation testing and validation platform for open-source 5g stack. In *2025 IEEE 30th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 1–2. IEEE, 2025.

[45] Linqi Xiao, Venkateswarlu Gudepu, and Andrea Fumagalli. Joint alarm correlation and root alarm detection in optical transport networks with graph-attention. In *2025 IEEE 30th International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*, pages 1–6. IEEE, 2025.

[46] Linqi Xiao, Aparaajitha Gomathinayakam Latha, Balagangadhar Bathula, Tianliang Zhang, and Andrea Fumagalli. Optical network robotic automation platform for multi-vendor optical transport network: accelerating validation, troubleshooting, and proof of concept efforts. In *2025 International Conference on Optical Network Design and Modeling (ONDM)*, pages 1–6. IEEE, 2025.

[47] Linqi Xiao, Abhishek Bhattacharyya, Moojan Kamalzadeh, Venkateswarlu Gudepu, Larry J Horner, and Andrea Fumagalli. Opening the black box: Zero-touch conformance and security testing of commercial o-rus. In *Proceedings of the 20th ACM Workshop on Wireless Network Testbeds, Experimental evaluation & Characterization*, pages 97–104, 2026.

[48] Linqi Xiao, Aparaajitha Gomathinayakam Latha, Venkateswarlu Gudepu, and Andrea Fumagalli. A yang-grounded llm agent supporting multi-vendor openroadm optical transport networks. In *Proceedings of the ACM SIGCOMM 2026 Conference*, pages 2109–2115, 2026.

[49] Siheng Xiong, Ali Payani, Yuan Yang, and Faramarz Fekri. Deliberate reasoning in language models as structure-aware planning with an accurate world model. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 31900–31931, 2025.

[50] Siheng Xiong, Ali Payani, and Faramarz Fekri. Enhancing language model reasoning with structured multi-level modeling. In *International Conference on Learning Representations*, volume 2026, pages 36557–36610, 2026.

[51] Siheng Xiong, Ali Payani, Oguzhan Gungordu, and Faramarz Fekri. Dive: Unlocking self-improvement in frozen language models through diversity-driven skill evolution. *arXiv preprint arXiv:2608.12486*, 2026.

[52] Junjielong Xu, Ying Fu, Shin Hwei Tan, and Pinjia He. Aligning the objective of llm-based program repair. In *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, pages 2548–2560. IEEE, 2025.

[53] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *Advances in Neural Information Processing Systems*, 37:50528–50652, 2024.

[54] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. *Advances in neural information processing systems*, 36:11809–11822, 2023.

[55] Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. Reevo: Large language models as hyper-heuristics with reflective evolution. *Advances in neural information processing systems*, 37:43571–43608, 2024.

[56] Yucheng Yuan, Yuanfeng Ji, Zhongxiao Li, and Ruijiang Li. Sp-mind: An autonomous reasoning agent for spatial proteomics analysis. *arXiv preprint arXiv:2606.24235*, 2026.

[57] Kechi Zhang, Zhuo Li, Jia Li, Ge Li, and Zhi Jin. 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)*, pages 769–787, 2023.

[58] Qizheng Zhang, Changran Hu, Shubhangi Upasani, Boyuan Ma, Fenglu Hong, Vamsidhar Kamanuru, Jay Rainton, Chen Wu, Mengmeng Ji, Hanchen Li, et al. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, volume 2026, pages 86069–86100, 2026.

[59] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1592–1604, 2024.

[60] Li Zhong, Zilong Wang, and Jingbo Shang. 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*, pages 851–870, 2024.

[61] Chunzheng Zhu, Yangfang Lin, Shen Chen, Yijun Wang, and Jianxin Lin. Medeyes: Learning dynamic visual focus for medical progressive diagnosis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 13916–13924, 2026.

[62] Chunzheng Zhu, Lei Tian, Bohan Tan, Ziqi Zhou, Yuxuan Sun, Yijun Wang, Chengchao Lv, Yilin Wen, Yijun He, Jinghao Lin, et al. The path to self-evolving clinical systems: Scaling medical agents from assistance to autonomy. *arXiv preprint arXiv:2607.11175*, 2026.

[63] Chunzheng Zhu, Yijun Wang, Jiaqi Zeng, Junyu Jiang, and Jianxin Lin. Medsynapse-v: Bridging visual perception and clinical intuition via latent memory evolution. In *European Conference on Computer Vision*, pages 559–577. Springer, 2026.

[64] Chunzheng Zhu, Jiaqi Zeng, Hongbo Zhao, Yihang Chen, Yijun Wang, and Jianxin Lin. Craft: Causal responsibility and failure tracing in medical vision language models. *arXiv preprint arXiv:2609.38810*, 2026.

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 4 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 4: 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 Experimental Protocol and Hyperparameters

Table 5 lists the settings shared by all formal comparisons in Section 4. The values are fixed across methods, tasks, and seeds and were not tuned per run.

Table 5: Hyperparameters and protocol settings for the matched comparisons.

Quantity	**Value / rule**
Search	
Population size N	16
Initialization (control)	Evaluate the task template p_0, then fill to $N/2$ with diagnostic (probability 0.6) or text-only mutations of p_0; children below $f_{p_0} - \max(1, 0.5\lvert f_{p_0}\rvert)$ are logged but not inserted
Parent selection	Tournament of size 3
Child insertion	Every valid, non-duplicate child is inserted and the N best are kept; invalid children are logged
UCB1 exploration constant C	1.2
UCB1 reward	Δ clipped to $[0, \Delta_{\max}]$ and divided by $\Delta_{\max}$, the running maximum of positive Δ; failed calls give 0; control tasks also credit ancestors' operators with $0.5^d r$, $d \leq 3$
Forced-burst trigger	12 consecutive non-improving committed children
Forced burst	One `m_inspire` and one `i1` proposal (control tasks)
Repair and revision	One repair call per crashing child; one revision call when a child of a parent within 5% of the best regresses by 2–50% (relative to $\max(1, \lvert f_p\rvert)$)
Skill retrieval	BM25 ($k_1 = 1.5$, $b = 0.75$), top $K = 3$; query = failure modes, root cause, and fix type of D_p
Skill utility	Initial 1; $+\min(\Delta, 1)$ if $\Delta > 0$, else $-\frac{1}{2}\min(\lvert\Delta\rvert, 1)$, for retrieved skills the Actor reports using
Skill archiving	$u_i < -1$ after ≥ 3 retrievals; LLM merge when token Jaccard > 0.85 and same target function; lowest utility first above capacity
Skill Memory capacity	256 skills
Diagnosis reuse	At most 2 Actor samples per Critic diagnosis
LLM backend and accounting	
Model weights (control comparisons)	Qwen/Qwen3-VL-235B-A22B-Instruct for every LLM role (Critic, Actor, repair, abstraction) in REFLEX, Actor-only, and official MLES
Model weights (transfer study)	Qwen3.5-27B
Output cap	4,096 tokens per call
Sampling workers	1
Budget accounting	Every successful or failed LLM generation attempt counts as one call; a diagnostic mutation therefore costs at least two calls (one Critic, one Actor)
Evaluation	
Campaign seeds	0–9 for all formal comparisons (paired across methods)
Control budgets	50 and 100 total LLM calls; 20 calls for the tight-budget, memory-object, and transfer studies
MLES baseline	Official implementation at commit `acda6457`, with native prompts, operators, selection, and duplicate handling
CCAA budget	240 objective evaluations per run, including REFLEX's inner local-search evaluations
CCAA initialization	Random design vectors, or a shared prior (the same first LLM/-analytical designs for every method)
Transfer banks	Top-2 skills by source-task utility, frozen and hash-verified, audited for target-specific identifiers
Statistics	95% paired-bootstrap CIs (20,000 resamples for the BE-perturbation study), exact Wilcoxon signed-rank tests, Holm correction within each planned family, Cohen's d_z

D 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 $8\times$ NVIDIA GeForce RTX 3090 GPUs. However, because the LLMs were queried through remote inference endpoints (Qwen/Qwen3-VL-235B-A22B-Instruct for all roles in the matched comparisons; Qwen3.5-27B in the transfer study), 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 dominated by inference latency rather than local compute. Throughout the paper, the Critic call and the Actor call of a diagnostic mutation are counted as two separate LLM calls (Table 5); together they take approximately 3 to 5 seconds on average. For a control-task budget of 100 total LLM calls, 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.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We clearly state our problem scope and contributions in Sec. 1

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Yes, please see Sec. 5 for limitations.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

Justification: The paper does not provide any theoretical results.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The method and algorithm are specified in Section 3 and Algorithm 1, the experimental protocol in Section 4, and all hyperparameters in Appendix C. The Critic and Actor prompt templates are given in Appendix A and example diagnoses in Appendix B; the prompts for the remaining operators are in the released code.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The source code, task evaluators, and commands to reproduce the control and CCAA experiments are publicly available at `https://github.com/wangpan-ustc/REFLEX` (linked in the abstract); all tasks are simulators, so no external dataset is required.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: As a train-free evolutionary framework, our method does not involve traditional neural network training (e.g., data splits or optimizers). However, we specify all evolutionary hyperparameters, LLM-call budgets, and the MLES and Actor-only configurations in Section 4 and Appendix C (fixed across methods, tasks, and seeds, not tuned per run); the settings of the CCAA classical optimizers are given in the released code.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [Yes]

Justification: Over 10 paired campaign seeds, we report mean $\pm$ SEM (Tables 1, 2, and 3; exceptions are marked in the captions) and median [IQR] calls or evaluations to target, together with 95% paired-bootstrap CIs, exact Wilcoxon signed-rank tests with Holm correction, and Cohen's d_z (Section 4, Appendix C).

Guidelines:

- The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Appendix D reports the hardware, the remote LLM inference setup, and the approximate per-call and per-campaign wall-clock time.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Yes, the research conducted in the paper conforms in every respect with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A]

Justification: NA.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: NA.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All third-party assets used in this work are properly cited and credited.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.

- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [N/A]

    Justification: NA.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: NA.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: NA.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.

- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: Large Language Models are the core operational components of our REFLEX framework.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.