GAE: Graph-Augmented Evolution for Scientific Discovery via Reinforcement Optimization

Anonymous Authors[1]

Abstract

Evolutionary program search guided by Large Language Models (LLMs) has emerged as a powerful paradigm for automated scientific discovery. However, current approaches are fundamentally constrained by three bottlenecks: structurally blind parent selection, sparse whole-program evaluation rewards, and static mutation operators that fail to adapt during search. We present **GAE** (**G**raph-**A**ugmented **E**volution), a framework that resolves these limitations through a tightly coupled, three-pillar architecture. First, a **relational graph neural network (GNN)** parses programs into typed computation graphs, producing structure-aware embeddings. Second, an **RL-optimized meta-controller** leverages these embeddings to replace blind evolutionary sampling with a directed policy, dynamically selecting optimal parents and mutation directions based on reward history. Third, an **online GRPO fine-tuning loop** continuously updates the LLM mutation operator at test-time using group-normalized evaluation rewards, directly aligning the model's generation distribution with high-fitness structural edits. We evaluate GAE on a challenging scientific discovery task: symbolic regression for complex nonlinear oscillator systems. By transforming stochastic search into a directed, self-improving trajectory, GAE efficiently discovers closed-form physical equations, consistently matching or outperforming static LLM-driven baselines and achieving state-of-the-art out-of-distribution performance.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

1. Introduction

The discovery of new algorithms and programs is a central frontier in AI-assisted science. Recent work follows a simple but powerful recipe: combine a large language model with programmatic evaluation and iterative search. FunSearch (Romera-Paredes et al., 2024) first demonstrated that this loop can discover genuinely new mathematical objects, outperforming decades of hand-crafted combinatorial search. AlphaEvolve (Novikov et al., 2025) generalized the principle to complete codebases, achieving state-of-the-art results on hardware scheduling, matrix multiplication, and open problems in math and engineering. Following this line of work, the overall procedure can be viewed as an iterative search framework in which an LLM serves as a program mutation operator that generates candidate code modifications, a task-specific evaluator assigns fitness based on predefined scoring criteria, and a selection mechanism iteratively preserves and refines high-performing programs within the search space.

Despite these successes, three interrelated bottlenecks hinder effective evolutionary search, especially in reliably steering program mutations toward high-quality regions of a large and complex program space. **(1) Reward sparsity.** Each fitness evaluation requires a complete program execution—a full training run in the case of neural architecture search, or a numerical integration in symbolic regression. The archive is updated only when a child outperforms the current occupant of its MAP-Elites cell, so the search process receives a single scalar signal indicating whether the candidate improves upon the incumbent in terms of the defined fitness score. Programs that nearly beat an incumbent, for example achieving performance very close to the current best, or that contain partially correct or structurally meaningful sub-expressions, are still discarded if they do not strictly improve the cell. As a result, a large population of evaluated programs is reduced to a sparse sequence of binary improvement events, discarding fine-grained relational information about proximity to improvement and the contribution of intermediate structural modifications. **(2) Uninformed parent selection.** AlphaEvolve samples parents by fitness rank with random diversity, treating the population as an unstructured bag of programs. However, the program population may

carry substantial structural information. Effective program evolution depends on the existence of local structural similarity in the program space. rather than merely high-level fitness comparison. Without such locality, small mutations induce unpredictable changes in program behavior, turning the search process into a largely stochastic exploration. In contrast, when programs exhibit meaningful local similarity, neighboring programs tend to induce more consistent and gradual changes in functionality, allowing iterative mutations to progressively refine performance. **(3) Static mutation operator.** In AlphaEvolve, the LLM for mutation is fixed throughout the entire search process, which constrains the space of program edits to a static and pre-trained distribution. As a result, the system continues to sample from the same set of structural transformations regardless of the accumulated search experience.

Our contribution. We present **G**raph-**A**ugmeted **E**volution, a framework that simultaneously resolves all three limitations within the OpenEvolve backbone. A **relational GNN** parses each program into a typed computation graph and is trained online to predict fitness (§3.3). A **Discrete SAC meta-controller** takes each program's GNN embedding and learns which structural edit to apply next, balancing task-score improvement against population novelty and complexity via automatic entropy regularization (§3.4). A **GRPO fine-tuning loop** uses group-normalized evaluation rewards and the parent's GNN embedding as a variance-reduction baseline to update the LLM mutation operator online, progressively sharpening mutation quality throughout search (§3.5). We evaluate GAE on an important scientific task—symbolic regression for nonlinear oscillator systems. We utilize the benchmark datasets from LLM-SR (Shojaee et al., 2024) for our experiments. In LLM-SR (§4) experiment, GAE discovers novel programs that consistently match or surpass prior baselines on the benchmarks, achieving competitive or state-of-the-art performance. The remainder of the paper is organized as follows: §2 surveys related work, §3 describes the GAE framework in full, §4 presents experimental results, §5 discusses limitations and future directions.

2. Related Work

Prior work addresses at most one of the above-mentioned bottlenecks in isolation. EvoTune (Surina et al., 2025) and ThetaEvolve (Wang et al., 2025) finetune the LLM via DPO and GRPO respectively, yet leave parent sampling uninformed. Surrogate-based optimization (White et al., 2021; Falkner et al., 2018) improves evaluation efficiency but typically operates over fixed, hand-encoded representations and does not support open-ended code evolution. Quality-Diversity RL methods (Nilsson and Cully, 2021; Faldor et al., 2023; Batra et al., 2023) replace random selection with policy-gradient updates, but require dense per-step rewards unavailable when fitness arrives only at program termination. No existing method closes all three gaps simultaneously.

GAE builds on LLM-driven evolutionary search but departs from its static operator assumption. FunSearch (Romera-Paredes et al., 2024), AlphaEvolve (Novikov et al., 2025), and OpenEvolve (Sharma, 2025) all freeze model weights throughout search, relying on prompt diversity alone for exploration. EvoPrompting (Chen et al., 2023) extends this idea through few-shot prompting, yet still fixes the selection policy. PACEvolve (Yan et al., 2026) addresses scaffold-level failure modes, such as context pollution, mode collapse, and weak collaboration, through hand-crafted rules, but leaves the mutation model and selection policy frozen. ThetaEvolve (Wang et al., 2025) introduces test-time RL fine-tuning to improve mutation quality, yet ties the weight update to a single task-specific objective rather than a general reusable selector. ShinkaEvolve (Lange et al., 2025) improves sample efficiency via a bandit-based LLM ensemble and novelty-based rejection filtering, yet the selection policy remains static and prompt-driven with no learned component. Our work adds a learned selector and fine-tuning loop on top of this backbone, so both the *which-to-mutate* and *how-to-mutate* decisions improve over time.

Maintaining population diversity while improving selection is precisely the goal of quality-diversity RL, yet existing methods expose a fundamental mismatch with program evolution. MAP-Elites (Mouret and Clune, 2015) archives diverse elites but samples parents uniformly. Policy gradient extensions PGA-MAP-Elites (Nilsson and Cully, 2021), DCG-MAP-Elites (Faldor et al., 2023), DCRL-MAP-Elites (Faldor et al., 2025), and PPGA (Batra et al., 2023) improve this but all require dense per-step rewards absent in whole-program evaluation. AURORA (Grillotti and Cully, 2022) relaxes hand-specified feature axes by learning behavioural descriptors unsupervisedly. GAE adopts this spirit for archive indexing while bridging the dense-reward gap through graph-structured signals propagated across the population.

3. Methodology

3.1. Background: MAP-Elites and OpenEvolve

MAP-Elites. Let $\mathcal{X}$ be a program space and $\mathcal{B}$ a k-dimensional behaviour descriptor space binned into cells $\mathcal{C}$. MAP-Elites maintains an archive $\mathcal{A} : \mathcal{C} \to \mathcal{X} \cup \{\emptyset\}$ mapping each cell to its current elite program. The QD-score aggregates quality and coverage:

$$\mathrm{QD}(\mathcal{A}) = \sum_{c \in \mathcal{C}} f(\mathcal{A}[c]) \cdot \mathbf{1}[\mathcal{A}[c] \neq \emptyset].$$

The deterministic update rule inserts a new program β into $\mathcal{A}$ only if it strictly improves its cell's incumbent, guaranteeing QD is non-decreasing.

OpenEvolve. At each generation, OpenEvolve selects a parent program from $\mathcal{A}$, assembles an LLM prompt with the parent code and top-k elites, queries the LLM for a mutated child (via SEARCH/REPLACE diffs or full rewrites), evaluates the child with a user-supplied function, and updates $\mathcal{A}$. Multiple islands evolve in parallel with periodic migration.

3.2. Graph-Augmented Evolution

Graph-Augmented Evolution extends the OpenEvolve loop with three co-adaptive components, all underpinned by a shared relational GNN encoder (§3.3) that parses each program's abstract syntax tree into a typed computation graph and produces a structural embedding $z \in \mathbb{R}^{128}$. In RL-guided parent selection (§3.4), this embedding replaces uniform parent sampling with an ε-greedy Q-network that scores every archive candidate by its predicted score improvement $Q_\phi(z)$, concentrating mutations on structurally promising parents. Whenever the LLM proposes a group of G children, online GRPO fine-tuning (§3.5) uses the group-relative rewards to update the LLM weights in-place, adapting the mutation policy without interrupting evolution while leaving the archive and the GNN unchanged. The GNN encoder itself is updated separately via an MSE surrogate loss against observed rewards. The three components share no gradients and operate at different timescales, keeping each module independently replaceable. Proposed extension (§3.4): when a calibrated GNN surrogate with uncertainty estimates $(\mu_\beta, \sigma_\beta)$ is available, the Q-selector in RL can be upgraded to a Discrete SAC meta-controller with a principled three-component reward decomposition. Figure 1 illustrates the complete architecture, and Algorithm 1 summarizes one generation of the GAE loop.

Algorithm 1 GAE — One Generation

Require: Archive $\mathcal{A}$ (MAP-Elites grid of elite programs, each p stores code, score R_p); GNN encoder Φ : code $\to \mathbb{R}^{128}$; Q-network $Q_\theta : \mathbb{R}^{128} \to \mathbb{R}$; LLM policy π_{LLM}; GNN fine-tune buffer $\mathcal{L}$; counter n; group size G=8; fine-tune interval K=20; exploration rate ε

// Step 1: Select parent via ε-greedy Q-selection
$z_p \leftarrow \Phi(p)\ \forall p \in \mathcal{A}$
$\alpha \leftarrow \begin{cases} \text{uniform}(\mathcal{A}) & \text{w.p. } \varepsilon \\ \arg\max_{p \in \mathcal{A}} Q_\theta(z_p) & \text{otherwise} \end{cases}$
// Step 2: Generate a group of G children
$\{\beta_1, \ldots, \beta_G\} \sim \pi_{\text{LLM}}(\cdot \mid \alpha, \mathcal{A})$
$R_{\beta_i} \leftarrow \text{Eval}(\beta_i, \mathcal{D}), \quad i = 1, \ldots, G$
// Step 3: GRPO policy update on the group
$\Delta_i \leftarrow R_{\beta_i} - R_\alpha, \quad \hat{A}_i \leftarrow \dfrac{\Delta_i - \mu_\Delta}{\sigma_\Delta + \epsilon}$
Update π_{LLM} via PPO-clip + KL penalty using $\{\hat{A}_i\}_{i=1}^G$
// Step 4: Archive update and bandit feedback
for $i = 1, \ldots, G$ **do**
    $\mathcal{A}.\text{TryInsert}(\beta_i)$; $n \leftarrow n + 1$
    Push (z_α, Δ_i) to Q-network buffer; train Q_θ if buffer ≥ 32
    $\varepsilon \leftarrow \max(\varepsilon_{\min}, \varepsilon \cdot \gamma_\varepsilon)$
    $\mathcal{L}.\text{Append}(\text{graph}(\beta_i), R_{\beta_i})$
end for
// Step 5: Periodic GNN fine-tuning
if $n \bmod K = 0$ **and** $|\mathcal{L}| \geq 10$ **then**
    Fine-tune Φ on $\mathcal{L}$; $z_p \leftarrow \Phi(p)\ \forall p \in \mathcal{A}$
end if

3.3. Program Graphs and GNN Encoder

Graph representation. We parse each candidate program into a typed computation graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ using static Abstract Syntax Tree (AST) analysis. Any parse failure produces a single `unknown` node, guaranteeing a valid graph is always returned. Specifically for symbolic regression, nodes are mathematical operations (one-hot over F=17 types) and edges encode data-flow (6 typed edge kinds).

Relational GNN encoder. Each node carries F input features (one-hot token type, positional index, etc.). We embed each program graph into $z \in \mathbb{R}^d$ ($d = 128$) via a relational graph neural network. **Input projection.**

$$h^{(0)} = \text{GELU}(XW_{\text{in}}), \quad X \in \mathbb{R}^{N \times F},\ W_{\text{in}} \in \mathbb{R}^{F \times d}. \tag{1}$$

Relational message passing. We apply $L = 2$ layers of edge-type-conditioned convolution. For each relation $r \in \mathcal{E}$, a dedicated weight matrix $W_r \in \mathbb{R}^{h \times h}$ transforms source node features; messages are mean-aggregated at each destination and combined with a self-loop:

$$h_i^{(\ell+1)} = \text{LayerNorm}\Big(\text{GELU}\Big(W_{\text{self}}\, h_i^{(\ell)} + \sum_{r=1}^{|\mathcal{E}|} \frac{1}{|\mathcal{N}_r(i)|} \sum_{j \in \mathcal{N}_r(i)} W_r\, h_j^{(\ell)}\Big)\Big). \tag{2}$$

Global pooling. Mean and max pooling are concatenated and projected to the output representation z as:

$$z = \text{GELU}\Big(\text{LayerNorm}\Big([\bar{h};\, \hat{h}]\, W_{\text{out}}\Big)\Big), \tag{3}$$

$$\bar{h} = \tfrac{1}{N} \textstyle\sum_i h_i^{(L)},\ \hat{h} = \max_i h_i^{(L)}. \tag{4}$$

Pretraining and online fine-tuning. Before evolution begins, the encoder is pretrained on synthetic programs with structural proxy scores (feature coverage, operator diversity, complexity) , using a temporary scalar head that is discarded

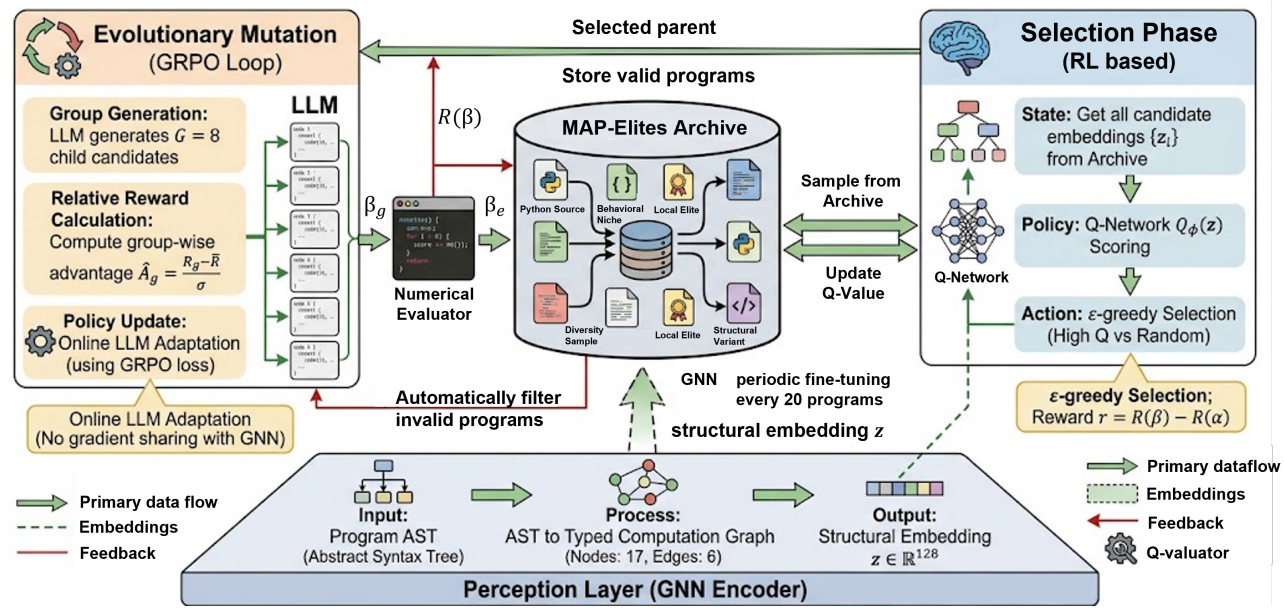

Figure 1. **GAE framework overview.** A relational GNN encoder parses each program's abstract syntax tree into a typed computation graph and produces a structural embedding, fine-tuned online against observed rewards to track the evolving program distribution. These embeddings drive an ϵ-greedy Q-network that replaces uniform parent sampling with learned, improvement-aware selection over the MAP-Elites archive. Each selected parent is mutated by a local LLM generating G=8 candidates, whose group-relative rewards are used to adapt the mutation policy via GRPO without altering the archive.

afterwards. During evolution, every 20 programs added to the database and the encoder is fine-tuned for 50 steps on a sliding window of the 300 most recent $(z,\ s)$ pairs from the program buffer $\mathcal{L}$, where $z \in \mathbb{R}^d$ is the GNN embedding of a program and $s \in \mathbb{R}$ is its evaluator score. All population embeddings are then refreshed. The training objective is:

$$\mathcal{L}_{\text{GNN}} = \frac{1}{|\mathcal{L}|} \sum_{\alpha \in \mathcal{L}} (\mu_\alpha - R(\alpha))^2, \tag{5}$$

where $\mu_\alpha = \text{MLP}(z_\alpha)$ is a scalar score prediction from a temporary head trained jointly with the encoder.

3.4. RL-Guided Parent Selection

Q-network. We learn a Q-function $Q_\phi : \mathbb{R}^d \to \mathbb{R}$ that predicts the expected score improvement for a candidate parent given its graph embedding z. The architecture is a three-layer MLP with LayerNorm and GELU activations:

$$Q_\phi(z) = \big(z \xrightarrow{128} \text{LayerNorm} \to \text{GELU} \xrightarrow{64} \text{GELU} \to \mathbb{R}\big).$$

ε-greedy selection. With probability ε , a parent is drawn uniformly at random from the current island; otherwise the parent maximising $Q_\phi(z_p)$ is selected. The selector operates in pure-random mode until the replay buffer contains at least 16 samples.

Reward design. The primary reward is the observed score improvement:

$$\Delta = R(\beta) - R(\alpha), \tag{6}$$

where α is the selected parent and β the evaluated child. To additionally encourage structural diversity and penalize runaway complexity, we define the following rewards respectively:

$$\begin{aligned} r_t^{\text{nov}} &= \lambda_1 \cdot \Big(1 - \min_{p \in \mathcal{P}} \cos(z_\beta, z_p)\Big), \\ r_t^{\text{cplx}} &= \lambda_2 \cdot \max\Big(0,\ \tfrac{N(\beta)-20}{20}\Big), \end{aligned} \tag{7}$$

where $\mathcal{P}$ is the current population, $N(\beta)$ is the AST node count of β's return expression, $\lambda_1 = 0.1$, and $\lambda_2 = 0.05$. The composite reward used by the standalone SAC evolution loop is:

$$r_t^{\text{tot}} = \text{clip}(R(\beta),\ r_{\min},\ r_{\max}) + r_t^{\text{nov}} - r_t^{\text{cplx}}, \tag{8}$$

with $r_{\min} = -5$ and $r_{\max} = 15$. The bandit Q-selector is trained on Δ alone (Eq. (6)).

Online Q-learning. After each evaluation we store (z_α, Δ) with $\Delta = R(\beta) - R(\alpha)$ in a replay buffer $(\mathcal{B})$ of capacity 500. Every 8 updates, a mini-batch of 32 pairs is sampled, targets are standardized across the batch, and one gradient step is taken via MSE loss with gradient clipping at 1.0:

$$\mathcal{L}(\phi) = \mathbb{E}_{(z,\Delta)\sim\mathcal{B}}\left[\Big(Q_\phi(z) - \tilde{\Delta}\Big)^2\right], \quad \tilde{\Delta} = \frac{\Delta - \bar{\Delta}}{\hat{\sigma}_\Delta + \epsilon}. \tag{9}$$

Discrete SAC meta-controller. When the GNN surrogate produces calibrated uncertainty σ_β, the bandit can be upgraded to a Discrete SAC with a factored policy $\pi_{\text{SAC}}(a_t|s_t) = \pi^{\text{sel}}_{\text{SAC}}(a^{\text{sel}}_t|s_t)\cdot\pi^{\text{dir}}_{\text{SAC}}(a^{\text{dir}}_t|s_t, a^{\text{sel}}_t)$ that jointly selects an archive cell and one of M semantic direction hints. This controller is trained on the following three-component reward:

$$r^{\text{tot}}_t = \underbrace{\big(\mu_\beta - \max_{\mathcal{A}[\text{cell}(\beta)]}\mu\big) - \big(\mu_\alpha - \max_{\mathcal{A}[\text{cell}(\alpha)]}\mu\big)}_{r^{\text{pot}}_t\text{: dense, policy-invariant}} + \underbrace{\lambda_u(t)\,\sigma_\beta}_{r^{\text{exp}}_t\text{: annealing exploration}} + \underbrace{\big(R(\beta) - \mu_\beta\big)\mathbf{1}[\text{eval}(\beta)]}_{r^{\text{cal}}_t\text{: sparse calibration}}. \tag{10}$$

The potential-based term $r^{\text{pot}}_t = \Phi(\beta) - \Phi(\alpha)$ with $\Phi(\alpha) = \mu_\alpha - \max_{\mathcal{A}[\text{cell}(\alpha)]}\mu$ preserves the optimal policy while providing a dense signal at every generation. The exploration bonus r^{exp}_t uses $\lambda_u(t) = \lambda_0 e^{-t/\tau_u}$ (λ_0=0.3, τ_u=$G/3$) to anneal UCB-style exploration. The calibration term r^{cal}_t corrects the surrogate prediction error at true evaluation steps. We leave the full SAC integration to future work.

3.5. GRPO LLM Fine-Tuning

Every two programs added, we construct a group of $G = 8$ children from the same parent: $\beta_g \sim \pi_{\theta_{\text{LLM}}}(\cdot|\alpha^*)$, $g = 1, \ldots, G$. All G children are truly evaluated, yielding rewards $\{R_g\}_{g=1}^G$. The group reward is baseline-corrected by the GNN surrogate score μ_{α^*} of the parent:

$$R^{\text{grp}}_g = R_g - \mu_{\alpha^*}, \quad \hat{A}_g = \frac{R^{\text{grp}}_g - \bar{R}^{\text{grp}}}{\text{std}(R^{\text{grp}}) + \epsilon}. \tag{11}$$

If $\text{std}(R^{\text{grp}}) < 10^{-8}$ (all rewards identical), the update is skipped. The GRPO objective fine-tunes θ_{LLM} with a PPO-clip surrogate and KL penalty $\beta_{\text{kl}} = 0.1$ to the reference model:

$$\mathcal{L}_{\text{GRPO}}(\theta_{\text{LLM}}) = -\frac{1}{G}\sum_g \min\big(\rho_g \hat{A}_g, \text{clip}(\rho_g, 1 \pm \epsilon_{\text{clip}})\hat{A}_g\big) + \beta_{\text{kl}} D_{\text{KL}}(\pi_{\theta_{\text{LLM}}} \| \pi^{\text{ref}}_{\theta_{\text{LLM}}}), \tag{12}$$

where $\rho_g = \exp\big(\log \pi_{\theta_{\text{LLM}}}(\beta_g|\alpha^*) - \log \pi^{\text{ref}}_{\theta_{\text{LLM}}}(\beta_g|\alpha^*)\big)$ and $\epsilon_{\text{clip}} = 0.2$. Completions with reward below -10 (evaluator crash sentinels) are dropped before computing advantages. When using LoRA, the reference policy is obtained by calling a specific function which temporarily masks the adapter weights and exposes the frozen base model, eliminating the need to load a separate reference model in memory. The best child (highest R_g) is added to the MAP-Elites database after each group.

4. Experiments

To evaluate GAEvolve's capability of scientific discovery, we test its performance on the Nonlinear Oscillators task from LLM-SR. Nonlinear damped oscillators are described by differential equations that capture the complex interaction among an oscillator's position, velocity, and the acting forces. The goal is to discover a closed-form expression $\hat{f}(x, t, v; \theta)$ that predicts the acceleration $\ddot{x} = dv/dt$ of a nonlinear harmonic oscillator, where x is position, t is time, v is velocity. We evaluate GAEvolve on discovering the governing equation of a synthetically generated variant by minimising the Normalised Mean Squared Error (NMSE) between predicted and observed trajectories. The actual evolved program expresses a candidate symbolic equation as executable code. The initial program is a naive linear model $\hat{v} = p_0 x + p_1 t + p_2 v + p_3$, parsed into an symbolic regression expression tree via the graph builder, producing a 4-node graph with `add`, `mul`, `input_var`, and `param` nodes.

The training set is used exclusively for parameter optimization, and the held-out test set is used for fitness evaluation. The target is to minimize MSE against ground-truth acceleration samples during evolution, and the performance metrics are reported as median NMSE on held-out (in-distribution and OOD) test sets.

Experimental Setup The experimental GAE training details are as follows: The MAP-Elites descriptor is two dimensional (expression tree depth, Stage-2 NMSE on 5 held-out trajectories), providing a diversity axis between interpretable shallow expressions and potentially more accurate deeper ones. Parent selection follows a UCB scheme with β annealed from 2.0 to 0.3 over the course of evolution, balancing exploration of low-visit archive cells against exploitation of high-fitness regions. Each generation samples LLM-generated candidates ($n_{\text{children}} = 8$) in parallel. The population consists of 200 programs across 4 isolated islands, with periodic migration every 15 generations. The cascade has three stages: Stage 1 verifies syntactic validity via a single-trajectory rollout; Stage 2 evaluates NMSE on 5 trajectories; Stage 3 computes full NMSE over all test trajectories. We compare three configurations under identical iteration budgets (150 iterations).

Before committing any LLM-generated candidate to the expensive numerical optimization pipeline, we apply a lightweight four-stage validation chain that filters out malformed programs early. We (i) extract valid Python code from the raw LLM output, (ii) verify syntactic correctness, (iii) sandbox the candidate by stripping unsafe statements, and (iv) execute a fast smoke test on a small data subset to catch runtime errors. Candidates failing any stage are discarded immediately, preventing invalid programs from entering the evaluator or corrupting the reinforcement learning replay buffer.

We use Qwen3.5-35B-A3B (Team, 2026) as the LLM back-

Method	NMSE ↓
Non-LLM baseline	
uDSR (Landajuela et al., 2022)	−4.06
LLM-guided evolution	
LLM-SR	−4.80
AlphaEvolve	−7.11
CodeEvolve	−7.26
ShinkaEvolve	−6.35
PACEvolve-Single	−8.23
PACEvolve-Multi	−8.24

Table 1. Benchmark results on the LLM-SR Nonlinear Oscillators task (Yan et al., 2026), reported as $\log_{10}$ NMSE (↓). All LLM-guided baselines run for 1000 iterations over 10 independent runs.

Method	ID NMSE ↓	OOD NMSE ↓
PACEvolve-Single (Gemini)	−5.80	−6.10
OpenEvolve (Gemini)	−3.14	−3.26
OpenEvolve (Qwen)	−3.88	−3.94
GAE (ours)	**−7.24**	**−7.45**

Table 2. Comparison among PACEvolve, OpenEvolve and our method GAE on the Nonlinear Oscillators task under our evaluation setting (150 iterations, 1 run), reported as $\log_{10}$ NMSE (↓) on in-distribution (ID) and out-of-distribution (OOD) test sets.

bone. We fix the LLM weights for PACEvolve-Single and OpenEvolve and finetune it in GAE using LoRA where only the adapter weights are updated during the GRPO reinforcement learning period. The GRPO has a group size of $G = 8$, and uses a KL regularization coefficient of $\beta = 0.1$ together with a clipping range of $\varepsilon = 0.2$ to stabilize policy updates. Optimization is carried out using AdamW with a learning rate of 1×10^{-6}. Fine-tuning is performed in an online manner, with updates triggered at every generation after candidate evaluation.

Main Results. First, we present results from PACEvolve (Yan et al., 2026) under their standard high-budget evaluation protocol (1000 iterations, 10 runs), which compares against uDSR (Landajuela et al., 2022), LLM-SR, AlphaEvolve, ShinkaEvolve (Lange et al., 2025). Table 1 reflects the performance of existing methods under their originally reported compute regime and provides a reference for overall competitiveness in the Non-linear harmonic oscillator task. Second, we conduct a controlled, low-budget comparison under a fixed backbone setting, using either Qwen or Gemini 2.5 Flash as the shared base model. In GAE, the Qwen backbone is fine-tuned via GRPO, while remaining fixed throughout all other pipeline components. In this regime, we compare the `OpenEvolve` baseline, the strong prior method `PaceEvolve`, and our proposed method **GAE**, all run for 150 iterations with a single run to ensure identical computational budget. As shown in Table 2, **GAE** achieves *best* performance compared to both baselines,

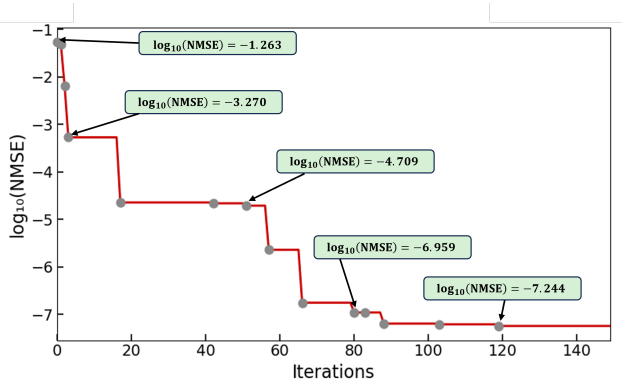

Figure 2. Performance trajectory of the GAE method on the Nonlinear Harmonic Oscillator symbolic regression problem. The curve shows the running-best $\log_{10}$(NMSE) as a function of the number of evolution iterations. Grey dots mark iterations at which a new best solution was discovered. Lower values of $\log_{10}$(NMSE) indicate better fit.

with its ID NMSE at -7.24 and OOD NMSE at -7.45 across all methods. Figure 2 shows the evolution trajectory of GAE over 150 iteration. The method starts from an initial solution with $\log_{10}(\text{NMSE}) = -1.26$ and improves rapidly during the first 20 iterations, reaching $\log_{10}(\text{NMSE}) \approx -4.64$ by iteration 17. Steady further gains accumulate through iterations 57–88, where three consecutive breakthroughs push the score to $\log_{10}(\text{NMSE}) = -7.19$. A final improvement at iteration 119 yields the best solution of the run, with $\log_{10}(\text{NMSE}) = -7.24$ (NMSE $\approx 5.7 \times 10^{-8}$), after which the trajectory plateaus for the remaining 30 iterations. Across all 150 iterations, 13 distinct improvements were recorded, demonstrating that GAE efficiently discovers a near-exact symbolic form of the target equation through graph-structured evolutionary search.

Next, we demonstrate the best program discovered by each of all four pipelines (Table 2) and provide physical interpretations of each component in the resulting equation.

GAE best program

$$\begin{aligned} f(x,t,v;\boldsymbol{\theta}) = {} & \theta_0\, x + \theta_1\, x^2\, \mathrm{sgn}(x) + \theta_2\, x^3 + \theta_3\, x^4\, \mathrm{sgn}(x) \\ & + \theta_4\, v + \theta_5\, v^2\, \mathrm{sgn}(v) + \theta_6\, v^3 \\ & + \theta_7\, \sin(t) + \theta_8\, \cos(t) \\ & + \theta_9\, \left(|x||v| + x^2|v| + |t||v| + t^2 v + x v^2\right) \end{aligned} \tag{13}$$

The terms $\theta_0 x$, $\theta_1 x^2\,\mathrm{sgn}(x)$, $\theta_2 x^3$, and $\theta_3 x^4\,\mathrm{sgn}(x)$ correspond to the linear restoring force, quadratic and cubic nonlinearity, and quartic nonlinearity respectively. The terms $\theta_4 v$, $\theta_5 v^2\,\mathrm{sgn}(v)$, and $\theta_6 v^3$ represent velocity-dependent damping. External time-dependent forcing is captured by $\theta_7 \sin(t)$ and $\theta_8 \cos(t)$. The five remaining terms $|x||v|$, $x^2|v|$, $|t||v|$, $t^2 v$, and $x v^2$, all scaled by the shared parame-

ter θ_9, are interaction terms.

OpenEvolve best program (Qwen)

$$\begin{aligned} f(x,t,v) = {} & \theta_1 x + \theta_2 x^3 + \theta_3 x^5 + \theta_4 v + \theta_5 v^3 \\ & + \theta_6 xv + \theta_7 x^2 v + \theta_8 x v^2 \\ & + \theta_9 \sin t + \theta_{10} \sin(2t) + \theta_{11} \cos t \\ & + \theta_{12} x \sin t + \theta_{13} v \sin t \\ & + \theta_{14} t + \theta_{15} t^2 + \theta_{16} t^3 \\ & + \theta_{17} \frac{1}{1+x^2} + \theta_{18} \frac{1}{1+v^2} \\ & + \theta_{19} e^{-x^2} + \theta_{20} e^{-v^2} \\ & + \theta_{21} \sinh(x) + \theta_{22} \cosh(x) \\ & + \theta_{23} x^2 t + \theta_{24} v^2 t + \theta_{25} xvt. \end{aligned}$$

The first three terms ($\theta_0 x$, $\theta_1 x^3$, $\theta_2 x^5$) represent a linear, cubic (Duffing-like), and quintic restoring force. The next two ($\theta_3 v$, $\theta_4 v^3$) capture linear and cubic damping. The mixed terms $\theta_5 xv$, $\theta_6 x^2 v$, and $\theta_7 x v^2$ are mixed linear damping with position, mixed cubic damping ($x^2 v$), and mixed cubic damping (xv^2). Harmonic, higher harmonic, and cosine driving are handled by $\theta_8 \sin(t)$, $\theta_9 \sin(2t)$, and $\theta_{10} \cos(t)$, while $\theta_{11} x \sin(t)$ and $\theta_{12} v \sin(t)$ introduce combined time-position and time-velocity interactions. The terms $\theta_{13} t$, $\theta_{14} t^2$, $\theta_{15} t^3$ are polynomial time terms. The terms $\theta_{16}/(1+x^2)$ and $\theta_{17}/(1+v^2)$ are inverse terms for potential singularities. The terms $\theta_{18} e^{-x^2}$ and $\theta_{19} e^{-v^2}$ model exponential decay or growth in position and velocity, and $\theta_{20} \sinh(x)$ and $\theta_{21} \cosh(x)$ provide smooth hyperbolic transitions. Finally, $\theta_{22} x^2 t$, $\theta_{23} v^2 t$, and $\theta_{24} xvt$ are cross-terms.

OpenEvolve best program (Gemini)

$$\begin{aligned} f(x,t,v;\boldsymbol{\theta}) = {} & \theta_0\, x + \theta_1\, \sin(t) + \theta_2\, v + \theta_3\, x^2 v \\ & + \theta_4\, x^2 + \theta_5\, v|v| + \theta_6\, x v^2 + \theta_7\, x^3 \\ & + \theta_8\, \cos(t) + \theta_9\, v^3 \end{aligned} \tag{14}$$

The linear term $\theta_0 x$ represents a linear restoring force, while $\theta_1 \sin(t)$ and $\theta_8 \cos(t)$ together capture periodic driving or intrinsic oscillation via sine and cosine components. The term $\theta_2 v$ encodes linear damping. The term $\theta_3 x^2 v$ is a higher-order interaction combining velocity-dependent nonlinear stiffness with position-dependent nonlinear damping. The quadratic term $\theta_4 x^2$ introduces an asymmetric potential or nonlinear restoring force. Rather than v^2, the term $\theta_5 v|v|$ is used to model quadratic drag, where the damping magnitude scales with v^2 but the sign always opposes the direction of motion—a physically accurate form of turbulent or aerodynamic damping. The term $\theta_6 x v^2$ is a higher-order nonlinear interaction between position and velocity. The cubic term $\theta_7 x^3$ introduces a Duffing-oscillator-like strong nonlinear restoring force, and $\theta_9 v^3$ adds cubic nonlinear damping.

PACEvolve Best Program

$$\begin{aligned} f(x,t,v;\boldsymbol{\theta}) = {} & \theta_0\, x + \theta_1\, v^3 + \theta_2\, v + \theta_3\, x\, e^{-v^2} + \theta_4\, x^3 \\ & + \theta_5\, v\, e^{-v^2} + \theta_6\, x v^2 + \theta_7\, \sin(t) \\ & + \theta_8\, \cos(t) + \theta_9\, x|x| \end{aligned} \tag{15}$$

The linear term $\theta_0 x$ represents a linear restoring force, $\theta_2 v$ encodes linear damping, and $\theta_1 v^3$ introduces cubic nonlinear damping. The cross-term $\theta_3 x\, e^{-v^2}$ is a nonlinear coupling between position and velocity, where the Gaussian envelope e^{-v^2} localizes the interaction to low-velocity regimes. The term $\theta_4 x^3$ adds a nonlinear restoring force of the Duffing-oscillator type. The term $\theta_5 v\, e^{-v^2}$ is a nonlinear damping contribution similarly localized by the Gaussian factor, attenuating its effect at large velocities. The higher-order cross-term $\theta_6 x v^2$ captures nonlinear coupling between position and the square of velocity. The terms $\theta_7 \sin(t)$ and $\theta_8 \cos(t)$ represent external periodic driving forces via sine and cosine components. Finally, $\theta_9 x|x|$ provides an additional nonlinear restoring force whose magnitude scales with x^2 but whose sign tracks the direction of displacement, analogous to quadratic drag in the position domain.

5. Conclusion

We presented **GAE**, a framework that augments LLM-guided MAP-Elites evolutionary search with three tightly coupled components: a relational GNN encoder that maps typed program graphs into structural embeddings, a contextual bandit selector that learns which program structures yield productive mutations, and a GRPO fine-tuning loop that adapts the mutation LLM online. Together, these components close three bottlenecks: reward sparsity, uninformed parent selection, and a static mutation operator. Experiments on symbolic regression show that GAE achieves the lowest NMSE both in-distribution and out-of-distribution among all baselines, attaining state-of-the-art performance.

Limitations. Our method relies on a graph representation derived from Python ASTs, which encodes domain-specific inductive biases. Extending to other languages or domains requires redesigning node types and relations, limiting out-of-the-box generalization and introducing non-trivial adaptation overhead. The benefit of graph-guided policy learning depends on a sufficiently expensive evaluation regime, as the GNN and reinforcement optimization introduce additional overhead.

Impact Statement

This work provides a fundamental methodological advancement in LLM-driven evolutionary algorithms. By introducing a dynamic, structure-aware paradigm—driven by relational graph encoders and online reinforcement optimization—our framework transforms evolutionary search from a largely stochastic process into a directed, self-improving trajectory. This algorithmic evolution is vital for scaling AI to tackle highly complex scientific problems, where the search spaces for viable mathematical or programmatic hypotheses are exceedingly vast and their evaluations computationally expensive.

The immediate impact of this framework is demonstrated in its ability to automate the discovery of interpretable, closed-form mathematical models from complex observational data. This methodology lays the technical groundwork for more comprehensive scientific agent systems designed to solve multifaceted scientific problems across theoretical physics, fluid dynamics, and complex systems engineering.

We foresee no significant harmful social consequences specific to this work. The broader social consequences of our work are those that are well established when advancing the fields of Machine Learning and AI for Science, and we do not feel any must be specifically highlighted here.

References

S. Batra, B. Tjanaka, M. C. Fontaine, A. Petrenko, S. Nikolaidis, and G. Sukhatme. Proximal policy gradient arborescence for quality diversity reinforcement learning. *arXiv preprint arXiv:2305.13795*, 2023.

A. Chen, D. M. Dohan, and D. R. So. Evoprompting: Language models for code-level neural architecture search, 2023. URL https://arxiv.org/abs/2302.14838.

M. Faldor, F. Chalumeau, M. Flageat, and A. Cully. Mapelites with descriptor-conditioned gradients and archive distillation into a single policy. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 138–146, 2023.

M. Faldor, F. Chalumeau, M. Flageat, and A. Cully. Synergizing quality-diversity with descriptor-conditioned reinforcement learning. *ACM Transactions on Evolutionary Learning*, 5(1):1–35, 2025.

S. Falkner, A. Klein, and F. Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International conference on machine learning*, pages 1437–1446. PMLR, 2018.

L. Grillotti and A. Cully. Unsupervised behavior discovery with quality-diversity optimization. *IEEE Transactions on Evolutionary Computation*, 26(6):1539–1552, 2022.

M. Landajuela, C. S. Lee, J. Yang, R. Glatt, C. P. Santiago, I. Aravena, T. Mundhenk, G. Mulcahy, and B. K. Petersen. A unified framework for deep symbolic regression. *Advances in Neural Information Processing Systems*, 35: 33985–33998, 2022.

R. T. Lange, Y. Imajuku, and E. Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025.

J.-B. Mouret and J. Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

O. Nilsson and A. Cully. Policy gradient assisted map-elites. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 866–875, 2021.

A. Novikov, N. Vũ, M. Eisenberger, E. Dupont, P.-S. Huang, A. Z. Wagner, S. Shirobokov, B. Kozlovskii, F. J. Ruiz, A. Mehrabian, et al. Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

B. Romera-Paredes, M. Barekatain, A. Novikov, M. Balog, M. P. Kumar, E. Dupont, F. J. Ruiz, J. S. Ellenberg, P. Wang, O. Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625 (7995):468–475, 2024.

A. Sharma. Openevolve: Open-source evolutionary code optimization with real-world gpu kernel discovery. *Hugging Face Blog*, 2025. URL https://huggingface.co/blog/codelion/openevolve.

P. Shojaee, K. Meidani, S. Gupta, A. B. Farimani, and C. K. Reddy. Llm-sr: Scientific equation discovery via programming with large language models. *arXiv preprint arXiv:2404.18400*, 2024.

A. Surina, A. Mansouri, L. Quaedvlieg, A. Seddas, M. Viazovska, E. Abbe, and C. Gulcehre. Algorithm discovery with llms: Evolutionary search meets reinforcement learning. *arXiv preprint arXiv:2504.05108*, 2025.

Q. Team. Qwen3.5 technical report. https://github.com/QwenLM/Qwen3.5, 2026. Accessed: 2026-05-08.

Y. Wang, S.-R. Su, Z. Zeng, E. Xu, L. Ren, X. Yang, Z. Huang, X. He, L. Ma, B. Peng, et al. Thetaevolve: Test-time learning on open problems. *arXiv preprint arXiv:2511.23473*, 2025.

C. White, W. Neiswanger, and Y. Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI conference*

on artificial intelligence, volume 35, pages 10293–10301, 2021.

M. Yan, B. Peng, B. Coleman, Z. Chen, Z. Xie, S. Chen, Z. He, N. Sachdeva, I. Ye, W. Wang, et al. Pacevolve: Enabling long-horizon progress-aware consistent evolution. *arXiv preprint arXiv:2601.10657*, 2026.

A. GNN Architecture Details

Input projection. For task $\tau \in \{\mathrm{SR}, \mathrm{MPP}\}$, the one-hot matrix $X \in \{0,1\}^{N \times |\mathcal{V}_\tau|}$ is projected to hidden dimension h: $h^{(0)} = \mathrm{GELU}(XW_{\mathrm{in}})$.

Relational convolution. For each relation $r \in \{1, \ldots, |\mathcal{E}|\}$, a dedicated $W_r \in \mathbb{R}^{h \times h}$ transforms source features; messages are mean-aggregated and summed with a self-loop W_{self}; each layer applies LayerNorm and GELU (Eq. 2). This is a *relational GCN*; no attention weights are computed.

Pretraining proxy scores. The proxy encodes a domain prior about what makes a program structurally promising, and is computed directly from graph statistics without any evaluation.

For SR, let $\mathcal{C}(G)$ be the set of distinct input-variable indices (`input_var` nodes), $\mathcal{F}(G)$ the set of distinct function-family node types (excluding atoms), and $|V|$ the total node count. The proxy score is:

$$s^{\mathrm{SR}}(G) = \underbrace{\frac{|\mathcal{C}(G)|}{3}}_{\text{input coverage}} + \underbrace{\frac{|\mathcal{F}(G)|}{5}}_{\text{operator diversity}} + \underbrace{\min\left(\tfrac{|V|}{15}, 1\right)}_{\text{complexity}} + p(|V|), \quad p(|V|) = \begin{cases} -0.5 & |V| \le 2 \\ -0.3 & |V| > 30 \\ 0 & \text{otherwise.} \end{cases} \tag{16}$$

Input coverage rewards equations that use all available physical quantities; operator diversity rewards varied mathematical structure over simple polynomials; and the complexity term favours moderate expression length.

Fallback. Any program failing AST parsing produces a single `unknown`/`unknown_layer` node; the encoder still produces a valid $z \in \mathbb{R}^{128}$.

B. Reward Design Comparison

Table 3 summarizes reward designs used in GAE, along with those found in QD-RL and LLM-RL research.

Table 3. Reward design comparison. **Bold**: GAE components.

Method	Source	Density	Scope	Recipient
PGA-MAP-Elites	r_t^{env}	Dense	Per step	TD3 critic
DCG-MAP-Elites	$r_t^{\mathrm{env}} - \|b_t - d^*\|$	Dense	Per step	Cond. critic
PPGA	r_t^{env} via GAE	Dense	Per step	PPO actor
EvoTune	Eval(x) pairwise	Sparse	Per program	LLM θ
ThetaEvolve	Eval$(x) - \lambda_{\mathrm{lazy}}$	Sparse	Per program	LLM θ
GAE (bandit)	$R(\beta) - R(\alpha)$ + novelty	Sparse	Per program	Q-selector
GAE (GRPO)	R_g^{grp} group-norm	Sparse	Per group	LLM θ
GAE (SAC, proposed)	$r^{\mathrm{pot}} + r^{\mathrm{exp}} + r^{\mathrm{cal}}$	Dense+sparse	Per gen	SAC

C. Decomposability of Bandit and GRPO

The bandit selector and GRPO optimise different parameter sets (θ_{sel} and θ_{LLM}) and their gradient flows never interact:

Component	Parameters	Reward	Without it
Bandit Q-selector	θ_{sel}	$\Delta = R(\beta) - R(\alpha)$	Uniform sampling
GRPO	θ_{LLM}	R_g^{grp} (group-norm)	Frozen LLM
GNN encoder	θ_{GNN}	MSE on $\mathcal{L}$	Random embeddings

D. Discrete Action Space — Expression Families

The proposed Discrete SAC meta-controller selects among four expression families at each generation, steering the LLM mutation prompt toward a structural region of the search space.

Table 4. Discrete action vocabulary for the SAC meta-controller (symbolic regression).

Action	Label	Description
0	`plain`	Simple or linear expressions
1	`polynomial`	Polynomial forms (e.g. x^2, x^3)
2	`nonlinear`	Smooth nonlinear functions (trigonometric, exponential, hyperbolic)
3	`interaction`	Cross-variable interaction terms

All hints are expressed in terms of the three observable inputs: position x, velocity v, and time t. Numeric constants are free parameters optimized by BFGS. Recommended primitives supplied to the LLM are `np.sin`, `np.cos`, `np.tanh`, `np.exp`, `np.abs`, `np.sign`, and `np.sqrt`.

E. Algorithm

Algorithm 2 Discrete SAC for Program Mutation

Require: GraphEncoder f_ψ, evaluator $\mathcal{E}$, replay buffer $\mathcal{B}$, task = SR
Initialise actor π_θ, critics Q_{ϕ_1}, Q_{ϕ_2}, targets, $\log \alpha$
for each episode **do**
  $s_0 \leftarrow f_\psi(\texttt{build\_graph}(c_0, \text{task}))$
  for $t = 0, 1, \ldots, T_{\max}$ **do**
    $a_t \sim \pi_\theta(\cdot \mid s_t)$
    $c_{t+1} \leftarrow \texttt{mutate}(c_t, a_t, \text{task})$ ▷ MutableSRTree or MutableMPPArchitecture
    score $\leftarrow \mathcal{E}(c_{t+1})$
    $s_{t+1} \leftarrow f_\psi(\texttt{build\_graph}(c_{t+1}, \text{task}))$
    Compute Δ via Eq. (6)
    $r_t \leftarrow \Delta + \lambda_n r_{\text{nov}} - \lambda_c r_{\text{cplx}}$ (Eq. 10)
    Store $(s_t, a_t, r_t, s_{t+1}, d_t)$ in $\mathcal{B}$
    Sample minibatch; update $Q_{\phi_1}, Q_{\phi_2}, \pi_\theta, \alpha$
    Soft-update targets with τ
    if d_t **then break**
    end if
  end for
end for