GAEVOLVE: GRAPH-AUGMENTED EVOLUTIONARY LLMS FOR SCIENTIFIC DISCOVERY VIA REINFORCEMENT OPTIMIZATION

Anonymous authors
Paper under double-blind review

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 **GAEvolve** (**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 GAEvolve on challenging scientific discovery tasks. By transforming stochastic search into a directed, self-improving trajectory, GAEvolve 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 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 as a full training run in the case of program 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.**

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

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 GAEvolve on an important scientific task—symbolic regression for nonlinear oscillator systems. In symbolic regression (§4.1) experiment, GAEvolve discovers novel programs that consistently match or surpass prior baselines on the benchmarks, achieving competitive or state-of-the-art performance. As a first step beyond equation discovery, we also apply GAEvolve to neural architecture search for molecular property regression (§4.3). The remainder of the paper is organized as follows: §2 surveys related work, §3 describes the GAEvolve framework in full, §4 presents experimental results, §5 concludes the work and discusses future directions.

2 Related Work

Prior work addresses at most one of the above-mentioned bottlenecks in isolation. EvoTune (Surina 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 & 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.

GAEvolve 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, 2025b) 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 & Clune, 2015) archives diverse elites but samples parents uniformly. Policy gradient extensions PGA-MAP-Elites (Nilsson & 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 & Cully, 2022) relaxes

hand-specified feature axes by learning behavioural descriptors unsupervisedly. GAEvolve 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. OpenEvolve maintains a program archive $\mathcal{A}$ as a quality–diversity population, using a MAP-Elites scheme in which programs are mapped onto a multi-dimensional feature grid and the best performer is retained per cell Sharma (2025a). At each generation, it samples a parent from $\mathcal{A}$, assembles an LLM prompt from the parent code and top-k elites (drawn separately from the parent, so inspiration programs differ from those shown to the LLM), and queries the LLM for a mutated child via SEARCH/REPLACE diffs or full rewrites. The child is scored by a user-supplied evaluator under a cascade pattern of multi-stage validation and admitted into $\mathcal{A}$ only if it dominates its cell's occupant. For diversity at scale, $\mathcal{A}$ is partitioned into multiple islands that evolve independently with periodic migration to prevent premature convergence, all orchestrated in parallel by a central controller.

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 $z \in \mathbb{R}^d$. Parents are drawn uniformly from a fixed-size elite archive; the shared embedding then conditions a discrete Soft Actor–Critic meta-controller (§3.4) that maps the parent state z to a mutation-type hint, steering the LLM toward a structurally promising class of edit rather than an undirected one. Its policy is trained on a three-component reward (task score, embedding-space novelty, and a complexity penalty) computed on the best child of each generation. Whenever the LLM proposes a group of G children, online GRPO fine-tuning (§3.5) converts their group-relative advantages $\hat{A}_i \propto R_{\beta_i} - R_\alpha$ (child reward minus parent reward) into an in-place PPO-clip update of the LLM weights, adapting the mutation policy without interrupting evolution while leaving the archive and the GNN unchanged. The GNN encoder itself is updated separately every K generations via an MSE surrogate loss that regresses its embeddings onto observed rewards, after which all archive embeddings are refreshed. The three components do not share gradients and operate at different timescales, keeping each module independently replaceable. Figure 1 illustrates the complete architecture, and Algorithm 1 summarizes one generation of the GAEvolve loop.

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 and edges encode data-flow.

Relational GNN encoder. Each node carries an F-dimensional one-hot encoding its node type. We embed each program graph into $z \in \mathbb{R}^d$ via a relational graph neural network. Suppose we denote the archive size as N, the **input projection is**

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

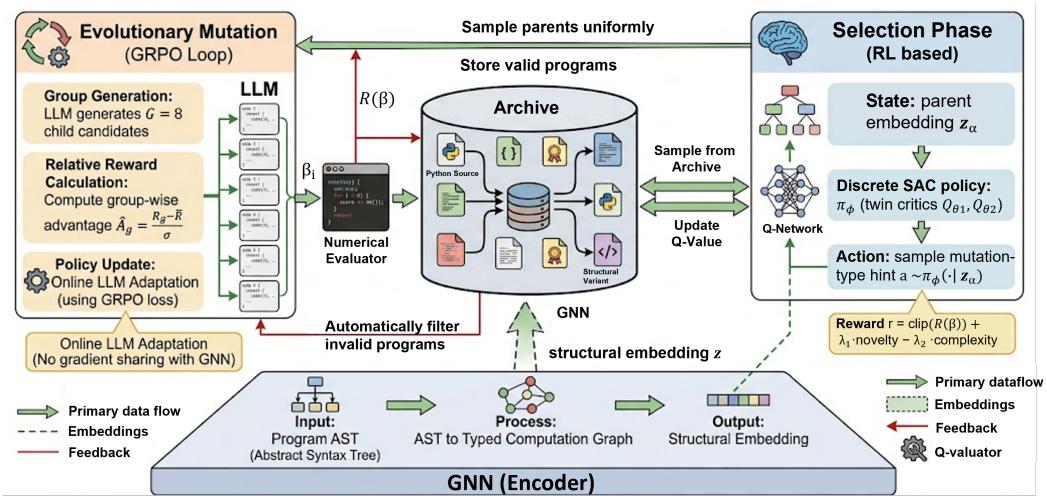


Figure 1: **GAEvolve framework overview.** A relational GNN encoder parses each program's abstract syntax tree into a typed computation graph and produces a structural embedding z, fine-tuned online against observed rewards to track the evolving program distribution. Parents are sampled from a fixed-size elite archive; the parent embedding z_α then serves as the state of a Discrete Soft Actor–Critic meta-controller (policy π_ϕ with twin critics $Q_{\theta_1}, Q_{\theta_2}$), which samples a mutation-type hint $a \sim \pi_\phi(\cdot \mid z_\alpha)$ that steers the mutation. Each parent is mutated by a local LLM generating $G = 8$ candidates, which are scored by a numerical evaluator. The SAC policy is trained on the best child of each generation with reward $r = \mathrm{clip}(R(\beta)) + \lambda_1 \cdot \text{novelty} - \lambda_2 \cdot \text{complexity}$, while the children's group-relative rewards adapt the LLM generation policy via GRPO.

Algorithm 1 GAEvolve — One Generation

Require: archive $\mathcal{P}$ (fixed size N); GNN encoder Φ : code $\to \mathbb{R}^d$ (d=128); SAC policy π_ϕ over action space $\mathcal{A}$ with twin critics $Q_{\theta_1}, Q_{\theta_2}$; LLM operator $\pi_{\theta_{\text{LLM}}}$ with reference π_{ref}; replay buffer $\mathcal{B}$; GNN window $\mathcal{L}$; group size G; GNN interval K; warm-up W; generation index s.

// Parent selection and mutation hint

$\alpha \sim \mathrm{Uniform}(\mathcal{P})$; $\quad z_\alpha \leftarrow \Phi(\alpha)$

$a \leftarrow \mathrm{Uniform}(\mathcal{A})$ if $s < W$, else $a \sim \pi_\phi(\cdot \mid z_\alpha)$

// Generate and evaluate a group of G children

$\{\beta_1, \ldots, \beta_G\} \sim \pi_{\theta_{\text{LLM}}}(\cdot \mid \alpha, a)$

$R_{\beta_i} \leftarrow \textsc{Eval}(\beta_i) = -\log_{10} \mathrm{NMSE}(\beta_i), \quad i = 1, \ldots, G$

// GRPO update of the LLM operator (§3.5)

$\hat{A}_i \leftarrow \left(R_i^{\text{grp}} - \overline{R^{\text{grp}}}\right) / (\mathrm{std}(R^{\text{grp}}) + \varepsilon), \quad R_i^{\text{grp}} = R_{\beta_i} - R_\alpha$

Update θ_{LLM} via the PPO-clip surrogate $\mathcal{L}_{\text{GRPO}}$ (Eq. 8)

// SAC meta-controller update, best child only (§3.4)

$\beta^\star \leftarrow \arg\max_i R_{\beta_i}$; $\quad z' \leftarrow \Phi(\beta^\star)$

$r \leftarrow \mathrm{clip}(R_{\beta^\star}, r_{\min}, r_{\max}) + \lambda_1 r_{\text{nov}}(z', \mathcal{P}) - \lambda_2 r_{\text{cx}}(\beta^\star)$ (Eq. 6)

$\mathcal{B}.\mathrm{push}(z_\alpha, a, r, z')$; update $\pi_\phi, Q_{\theta_1}, Q_{\theta_2}$ (Eq. 7) if $|\mathcal{B}| \geq 32$

// Archive and GNN-buffer update (all children)

for $i = 1, \ldots, G$ **do**

  $\mathcal{P}.\mathrm{Insert}(\beta_i, R_{\beta_i}, \Phi(\beta_i))$, keep top-$N$ by score

  append $(\mathrm{graph}(\beta_i), R_{\beta_i})$ to $\mathcal{L}$ if $R_{\beta_i} > -10$

end for

// Periodic GNN fine-tuning (§3.3)

if $s \bmod K = 0$ **and** $|\mathcal{L}| \geq 10$: fine-tune Φ (Eq. 5); refresh $z_p \; \forall p \in \mathcal{P}$

with hidden width h (the output dimension is produced only by the final pooling projection, where $2h = d$).

Relational message passing. We apply L 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 the features of the source node. Messages are mean-aggregated at each destination and combined with a self-loop:

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

where $\mathcal{N}_r(i) = \{\, j : (j,i) \in \mathcal{E}_r \,\}$ denotes the set of neighbors of node i connected by an edge of relation type r, and $|\mathcal{N}_r(i)|$ is its cardinality.

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

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

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

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 linear score-prediction head trained jointly with the encoder. During evolution, every 3 steps (once ≥ 10 samples exist), the encoder is fine-tuned for 300 gradient steps on a sliding window of the 300 most recent $(z_\alpha, R(\alpha))$ pairs, where $z_\alpha \in \mathbb{R}^d$ is the GNN embedding of program α and $R(\alpha) \in \mathbb{R}$ is its evaluator score. Targets are standardized to zero mean and unit variance. All population embeddings are then refreshed. The training objective is:

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

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

3.4 RL-Guided Mutation Selection

At each evolution step a parent α is drawn *uniformly at random* from the current island; the reinforcement-learning agent does not select parents. Instead, a Discrete Soft Actor-Critic (SAC) agent chooses which *mutation* to apply to α, conditioned on the parent's graph embedding.

State and action space. The state is the GNN embedding $z_\alpha \in \mathbb{R}^d$ of the parent program. The action space is flat and fixed, with $|\mathcal{A}|$ discrete actions, each encoding a tuple (mutation type, target node id, argument index).

Networks. The policy $\pi_\phi(a \mid z)$ and the twin critics $Q_{\theta_1}, Q_{\theta_2} : \mathbb{R}^d \to \mathbb{R}^{|\mathcal{A}|}$ are three-layer MLPs with LayerNorm and GELU activations that score every action from the state:

$$z \xrightarrow{256} \mathrm{LayerNorm} \to \mathrm{GELU} \xrightarrow{256} \mathrm{GELU} \to \mathbb{R}^{|\mathcal{A}|}.$$

Each critic is paired with a target network updated by Polyak averaging (coefficient τ).

Action selection. During a warm-up of W steps, actions are sampled uniformly at random to seed the replay buffer with diverse transitions. Thereafter, the agent samples an action stochastically from the softmax policy $\pi_\phi(a \mid z_\alpha)$ and no policy update is performed until the buffer holds at least 32 transitions.

Reward. The mutation producing the best child β receives a three-component reward:

$$r = \underbrace{\mathrm{clip}\big(R(\beta),\, r_{\min},\, r_{\max}\big)}_{r_{\mathrm{task}}} + \underbrace{\lambda_1 \big(1 - \min_{p \in \mathcal{P}} \cos(z_\beta, z_p)\big)}_{r_{\mathrm{novelty}}}$$

$$\underbrace{- \lambda_2\, \max\Big(0,\, \tfrac{N(\beta)-20}{20}\Big)}_{r_{\mathrm{complexity}}}, \tag{6}$$

where $R(\beta)$ is the evaluator score of the best child, $\mathcal{P}$ is the current population, $N(\beta)$ is the AST node count of β's return expression. The task term uses the child's absolute (clipped) score rather than a parent-relative improvement.

Online SAC training. Each transition $(z_\alpha, a, r, z_\beta)$ is stored in a prioritized replay buffer of capacity. Every generation we take K gradient steps (equal to the number of children per generation) on priority-sampled mini-batches, updating the twin critics, the policy, and the entropy temperature α (automatically tuned toward a target entropy of $-\frac{1}{2}\log|\mathcal{A}|$). The critics minimize an importance-weighted TD loss

$$\mathcal{L}(\theta_i) = \mathbb{E}_{(z,a,r,z')\sim\mathcal{B}}\Big[w\,\big(Q_{\theta_i}(z,a) - y\big)^2\Big],$$
$$y = r + \gamma\, V(z'),$$

with bootstrapped value

$$V(z') = \sum_a \pi_\phi(a \mid z')\big(\min_i Q_{\bar{\theta}_i}(z',a) - \alpha \log \pi_\phi(a \mid z')\big) \tag{7}$$

and importance weight w. The policy maximizes the standard entropy-regularized objective

$$J(\pi) = \mathbb{E}_{z\sim\mathcal{B}}\Bigg[\sum_a \pi_\phi(a \mid z)\big(\min_i Q_{\theta_i}(z,a) - \alpha \log \pi_\phi(a \mid z)\big).$$

3.5 GRPO LLM Fine-Tuning

On every evolution, we construct a group of G children from the sampled parent α, $\beta_g \sim \pi_{\theta_{\text{LLM}}}(\cdot \mid \alpha, a),\ g = 1, \ldots, G$, with a denoting the action selected by the SAC meta-controllor. A GRPO update is performed immediately after the group is evaluated. Every valid child is truly evaluated, yielding rewards R_g and malformed completions are discarded before evaluation.

The group reward is baseline-corrected by the parent's own (true) evaluated score R_α: $R_g^{\text{grp}} = R_g - R_\alpha$, i.e. each child's reward is measured as its improvement over the parent. 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:

$$\mathcal{L}_{\text{GRPO}}(\theta_{\text{LLM}}) = -\frac{1}{G}\sum_g \min\big(\rho_g \hat{A}_g,\ \text{clip}(\rho_g, 1 \pm \epsilon_{\text{clip}})\,\hat{A}_g\big) + \beta_{\text{kl}}\left[D_{\text{KL}}(\pi_{\theta_{\text{LLM}}} \,\|\, \pi^{\text{ref}}_{\theta_{\text{LLM}}})\right], \tag{8}$$

where $\rho_g = \exp\big(\log \pi_{\theta_{\text{LLM}}}(\beta_g|\alpha) - \log \pi^{\text{ref}}_{\theta_{\text{LLM}}}(\beta_g|\alpha)\big)$, ϵ_{clip} is PPO clipping range, and the group-normalized advantage of child g

$$\hat{A}_g = \frac{R_g^{\text{grp}} - \bar{R}^{\text{grp}}}{\text{std}(R^{\text{grp}}) + \epsilon}. \tag{9}$$

The KL term is therefore evaluated for monitoring only and contributes no gradient. The update of θ_{LLM} comes solely from the PPO-clip surrogate, with clipping to the trust region $[1-\epsilon_{\text{clip}},\ 1+\epsilon_{\text{clip}}]$ providing the effective regularization and β_{kl} reported for diagnostics.

Completions with reward below -10 (evaluator crash sentinels) are dropped before computing advantages. After each group, all evaluated children are inserted into the fixed-size elite population (a top-K list sorted by score, with the best child inserted first for elitism), and the population is truncated to its maximum size.

4 Experiments

4.1 Symbolic Regression

We first evaluate GAEvolve on equation discovery for nonlinear oscillators: the goal is a closed-form expression $\hat{f}(x, t, v; \theta)$ for the acceleration $\dot{v}$ of a nonlinear harmonic oscillator from position x, time t and velocity v, written as executable code. The seed program is the linear model $\hat{v} = p_0 x + p_1 t + p_2 v + p_3$.

Dataset. We use the `phys_osc` (nonlinear harmonic oscillator) domain of the LSR-Synth subset of LLM-SRBench (Shojaee et al., 2025), predicting the acceleration $\dot{v}$ from (x, t, v). Each task supplies three disjoint sets: a training set, an in-distribution (ID) test set, and an out-of-distribution (OOD) test set sampled from a different input time region. For every candidate equation, the numeric constants are optimized by BFGS to minimize the mean squared error (MSE) on the training set. The candidate is then scored by $-\log_{10}(\text{NMSE})$ on the held-out ID test set, which serves as the evolutionary fitness. We report $\log_{10}$ NMSE of the best program on the ID and OOD test sets; the OOD set is never used during search and therefore quantifies extrapolation. For every method, the reported best program is re-fitted with 20 random BFGS restarts on the training set before scoring.

Experimental Setup. GAEvolve maintains a fixed-size elite population of 50 programs sorted by fitness. Parents are sampled uniformly from this population, and a discrete Soft Actor-Critic meta-controller (with 32 random warm-up steps) selects the mutation-type hint passed to the LLM. Each generation samples eight LLM-generated candidates in parallel. 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 randomly generated dummy inputs 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.

For the OpenEvolve baseline, the MAP-Elites archive is two-dimensional (expression tree depth, number of distinct AST node types as structural diversity), with a population of 200 programs across 4 isolated islands and periodic migration every 15 generations. Parent sampling uses fixed elite/exploration/exploitation, each at the ratio of 0.3/0.4/0.3. For the PACEvolve baseline, We use the official implementation with a single island. The program database maintains a score-sorted queue of at most 100 programs, and each parent is drawn by tournament selection of size 2 from the top-4 programs. Each iteration generates one candidate with the frozen LLM backbone with temperature at 0.7 and top-p at 0.95). Before code generation, a scratch-pad step proposes a hypothesis, a repository capped at 5 ideas with periodic merging, and backtracking which restores an earlier repository state sampled from a power-law distribution.

We use Qwen3.5-35B-A3B Team (2026) as the LLM backbone of GAEvolve. We fix the LLM weights for PACEvolve-Single and OpenEvolve and update it in full weights in GAEvolve 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 Pro as the shared base model. In this regime, we compare the `OpenEvolve` baseline, the strong prior method `PaceEvolve`, and our proposed method **GAEvolve**, all run for 150 iterations to ensure identical computational budget. As shown in Table 2, **GAEvolve** achieves *best* performance compared to both baselines, with its ID NMSE at -7.24 and OOD NMSE at -7.45 across all methods.

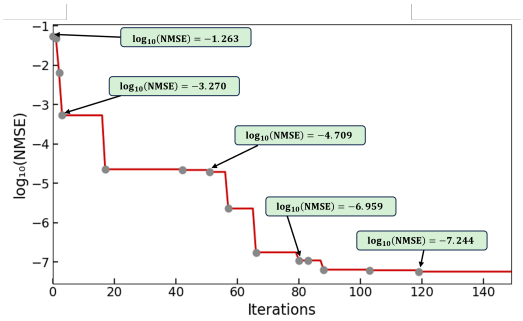
Figure 2a shows the evolution trajectory of GAEvolve over 150 iteration. The method starts from an initial solution with $\log_{10}(\text{NMSE}) = -1.26$ and improves rapidly during the first 20 iterations, reaching $\log_{10}(\text{NMSE}) \approx -4.64$ by iteration 17. Steady further gains accumulate 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 GAEvolve efficiently discovers a near-exact symbolic form of the target equation through graph-structured evolutionary search.

Method	Mean	Best
Non-LLM baseline		
uDSR	-3.95	-4.06
LLM-guided evolution		
LLM-SR	-4.06	-4.80
OpenEvolve	-5.40	-7.11
CodeEvolve	-4.97	-7.26
ShinkaEvolve	-5.35	-6.35
PACEvolve-Single	-5.87	-8.23
PACEvolve-Multi	-6.11	-8.24

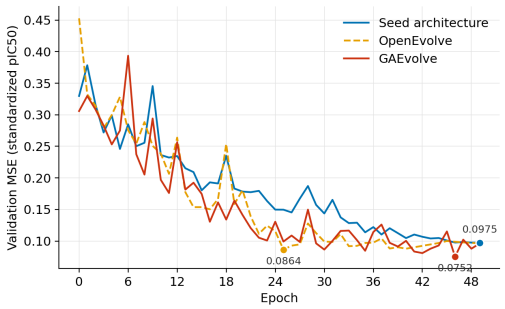
Table 1: Reference results reported by Yan et al. (2026) on the same task (1000 iterations, 10 runs, proprietary backbones), in $\log_{10}$ NMSE (↓).

Method	LLM	ID ↓	OOD ↓
PACEvolve-Single	Gemini	-5.11 ± 0.87	-5.26 ± 0.94
OpenEvolve	Gemini	-2.61 ± 0.94	-2.87 ± 0.59
OpenEvolve	Qwen	-5.70 ± 1.35	-5.77 ± 1.35
GAEvolve (ours)	Qwen	$\mathbf{-6.87} \pm 0.44$ **best** $\mathbf{-7.24}$	$\mathbf{-7.10} \pm 0.41$ **best** $\mathbf{-7.45}$

Table 2: Controlled comparison at 150 iterations on the nonlinear-oscillator task, $\log_{10}$ NMSE (↓), mean ± standard deviation over 3 runs.



(a) Symbolic regression: running-best $\log_{10}$ NMSE.



(b) pIC50: validation loss when retraining each best program.

Figure 2: (a) Search trajectory of the best GAEvolve run on the nonlinear-oscillator task; grey dots mark new best programs. The run was resumed from its iteration-100 checkpoint, so iterations 100–149 come from the resumed segment. (b) Validation MSE (standardized pIC50) while retraining the seed architecture and the best architectures found by OpenEvolve and GAEvolve; markers show each curve's minimum.

Search dynamics. We look into the GAEvolve search dynamics and how the learned components behave. Once the uniform warm-up steps end, the entropy of the SAC policy falls from 7.90 nats (uniform over $|\mathcal{A}| = 2700$) to about 3.6 nats within roughly 15 generations, drifts down to 3.1 nats by iteration 99 and stays there. The policy thus concentrates on a small set of edits: it never again requests `replace_operator`, which was sampled 5 times during warm-up without producing an improvement, and `add_operator` becomes the most frequent hint in iterations 32–89, accounting for 4 of the 7 improvements in that phase. The improvement is not confined to the running best. The median score of *all* generated children rises from 2.6 in the first third of the run to 4.7 in the second and 6.9 in the last, which is consistent with the GRPO-adapted LLM proposing better edits overall rather than occasionally getting lucky.

4.2 Ablation Studies

We evaluate the contributions of GRPO and SAC by removing each component independently. Table 3 reports the mean and population standard deviation over a few runs. Performance is measured using $\log_{10}(\text{NMSE})$ on both the in-distribution and out-of-distribution test sets, where lower values indicate better performance.

Removing SAC produces a lower average error than removing GRPO on both ID and OOD data. However, the No-SAC variant also exhibits substantially higher variance across runs. In contrast, the No-GRPO variant is more consistent but has worse average performance. These results suggest that

Table 3: Ablations on the nonlinear-oscillator task (150 iterations), mean $\pm$ standard deviation over a few runs, $\log_{10}(\text{NMSE})$ (↓).

Method	ID $\log_{10}(\text{NMSE})$	OOD $\log_{10}(\text{NMSE})$
GAEvolve (full)	-6.87 ± 0.44	-7.10 ± 0.41
No-GRPO (SAC only)	-5.98 ± 0.29	-6.04 ± 0.27
No-SAC (GRPO only)	-6.37 ± 0.69	-6.50 ± 0.76

GRPO has a meaningful effect on predictive accuracy, while SAC contributes to the stability of the optimization process.

4.3 Beyond Symbolic Regression: Architecture Search for Molecular Property Prediction

As a test of whether GAEvolve transfers to program spaces far larger than symbolic expressions, we apply it to neural architecture search for molecular bioactivity regression. Each program is a complete PyTorch model definition, together with its training hyperparameters, and each fitness evaluation is a training run.

We predict pIC50 for 15,037 molecules from a public SMILES collection (Kaggle *big-molecules-smiles*). The molecules are converted to RDKit 3D conformers in Uni-Mol format (Zhou et al.) (atom tokens, pairwise distances and pair types) and split randomly 80/20 into training and validation sets, with standardized targets. The seed program is a 4-layer molecular transformer (hidden size 128, 4 heads) with a CLS token and an attention bias computed from Gaussian distance kernels and pair types; all weights are trained from scratch. Candidates pass a cascade of a compile check and a 5-epoch training run before full training. Fitness combines validation R^2 and MAE with a small bonus for parameter efficiency. We compare against OpenEvolve with the same seed. The best program of each method is retrained for 50 epochs and compared by validation MSE. State the LLM, iteration budget and GAEvolve configuration used for these pIC50 runs, and the parameter counts of the evolved models.

Table 4: pIC50: best validation MSE (standardized) when retraining the best program (1 run).

Program	Val MSE ↓	Epoch
Seed transformer	0.0975	49
OpenEvolve	0.0864	25
GAEvolve	**0.0752**	46

GAEvolve's architecture reaches a validation MSE of 0.0752, 23% below the seed (0.0975) and 13% below the best OpenEvolve architecture (0.0864; Table 4, Figure 2b). The evolved program (Appendix F.2) deepens and widens the seed to 8 layers of width 384 with 12 heads. It also introduces gated Gaussian distance kernels with an additional learned distance-bin bias, pair representations updated from attention logits, gated feed-forward blocks with RMSNorm and scaled residuals, softmax-weighted aggregation over layers, and two pooled prediction heads whose outputs are averaged. Several of these components mirror design choices of Uni-Mol-style pretrained models, but here they were discovered from a small from-scratch seed.

5 Conclusion

We presented **GAEvolve**, which augments LLM-guided evolutionary search with three coupled components: a relational GNN encoder that maps typed program graphs to structural embeddings and is trained online on observed fitness; a discrete SAC meta-controller that selects the class of edit to request from the LLM based on the parent's structure; and a GRPO loop that adapts the mutation LLM online from parent-relative group rewards. On equation discovery for nonlinear oscillators, GAEvolve with an open-weight backbone obtains the best single-run and mean $\log_{10}$ NMSE both in and out of distribution among the evaluated methods at a low iteration budget. A preliminary application to neural architecture search for pIC50 prediction suggests that the same machinery transfers to much larger program spaces. Future work includes evaluation-matched comparisons, more runs, and further scientific domains.

AI USE STATEMENT

In this work, we used generative AI tools to provide feedback on research methodology and experimental design, implement methods, clean and reformat datasets, and support qualitative and thematic data analysis. We have not used generative AI tools to generate synthetic datasets, develop theoretical models or conceptual frameworks, propose or refine hypotheses, or interpret results; the formulation of mathematical claims, the provision of critical ingredients for proofs, the writing of proofs, and translation are not applicable to this work. Additionally, we used generative AI tools to create or modify scientific figures and to edit the paper for readability. We have reviewed all AI-assisted work: AI-generated code was verified and tested for correctness by the authors, and AI-assisted figures and text were checked against the underlying experimental records. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Sumeet Batra, Bryon Tjanaka, Matthew C Fontaine, Aleksei Petrenko, Stefanos Nikolaidis, and Gaurav Sukhatme. Proximal policy gradient arborescence for quality diversity reinforcement learning. *arXiv preprint arXiv:2305.13795*, 2023.

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

Maxence Faldor, Félix Chalumeau, Manon Flageat, and Antoine Cully. Map-elites with descriptor-conditioned gradients and archive distillation into a single policy. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 138–146, 2023.

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

Stefan Falkner, Aaron Klein, and Frank Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International conference on machine learning*, pp. 1437–1446. PMLR, 2018.

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

Mikel Landajuela, Chak Shing Lee, Jiachen Yang, Ruben Glatt, Claudio P Santiago, Ignacio Aravena, Terrell Mundhenk, Garrett Mulcahy, and Brenden K Petersen. A unified framework for deep symbolic regression. *Advances in Neural Information Processing Systems*, 35:33985–33998, 2022.

Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution. *arXiv preprint arXiv:2509.19349*, 2025.

Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

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

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.

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.

Asankhaya Sharma. Openevolve: an open-source evolutionary coding agent, 2025a. URL https://github.com/algorithmicsuperintelligence/openevolve.

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

Parshin Shojaee, Ngoc-Hieu Nguyen, Kazem Meidani, Amir Barati Farimani, Khoa D Doan, and Chandan K Reddy. Llm-srbench: A new benchmark for scientific equation discovery with large language models. *arXiv preprint arXiv:2504.10415*, 2025.

Anja Surina, Amin Mansouri, Lars Quaedvlieg, Amal Seddas, Maryna Viazovska, Emmanuel Abbe, and Caglar Gulcehre. Algorithm discovery with llms: Evolutionary search meets reinforcement learning. *arXiv preprint arXiv:2504.05108*, 2025.

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

Yiping Wang, Shao-Rong Su, Zhiyuan Zeng, Eva Xu, Liliang Ren, Xinyu Yang, Zeyi Huang, Xuehai He, Luyao Ma, Baolin Peng, et al. Thetaevolve: Test-time learning on open problems. *arXiv preprint arXiv:2511.23473*, 2025.

Colin White, Willie Neiswanger, and Yash Savani. Bananas: Bayesian optimization with neural architectures for neural architecture search. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pp. 10293–10301, 2021.

Minghao Yan, Bo Peng, Benjamin Coleman, Ziqi Chen, Zhouhang Xie, Shuo Chen, Zhankui He, Noveen Sachdeva, Isabella Ye, Weili Wang, et al. Pacevolve: Enabling long-horizon progress-aware consistent evolution. *arXiv preprint arXiv:2601.10657*, 2026.

Gengmo Zhou, Zhifeng Gao, Qiankun Ding, Hang Zheng, Hongteng Xu, Zhewei Wei, Linfeng Zhang, and Guolin Ke. Uni-mol: A universal 3d molecular representation learning framework. In *The eleventh international conference on learning representations*.

A GNN Architecture Details

Input projection. For task τ, 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 $r \in \{1,\dots,|\mathcal{E}|\}$, 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*, and no attention weights are computed.

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 Decomposability of SAC, GRPO, and the GNN Encoder

The SAC meta-controller, the GRPO fine-tuning loop, and the GNN encoder optimize disjoint parameter sets ($\{\phi, \theta_1, \theta_2\}$, θ_{LLM}, and θ_{GNN} respectively). No gradients flow between them, and each operates on its own timescale (per generation, per group, and every K generations). Disabling any one component degrades gracefully to a simpler baseline:

Component	Parameters	Training signal	Without it
SAC meta-controller	$\phi,\ \theta_1, \theta_2$	$r = \mathrm{clip}(R(\beta^\star)) + \lambda_1 r_{\mathrm{nov}} - \lambda_2 r_{\mathrm{cx}}$ (Eq. 6)	Uniform mutation hint
GRPO	θ_{LLM}	$\hat{A}_g$ from $R_g^{\mathrm{grp}} = R_g - R_\alpha$ (Eqs. 8–9)	Frozen LLM
GNN encoder	θ_{GNN}	$\mathcal{L}_{\mathrm{GNN}}$: MSE on $\mathcal{L}$ (Eq. 5)	Stale embeddings

Parent selection is uniform over the elite archive in all configurations and is unaffected by any of the three components.

C Discrete Action Space — Mutation Operators

The Discrete SAC meta-controller operates over a flat, fixed action space of $|\mathcal{A}| = 5\times30\times18 = 2700$ discrete actions, each encoding a tuple (*mutation type*, *target node id*, *argument index*). The target node id indexes the parent's expression tree in BFS order (capped at 30 nodes), and the argument index selects an operator or constant from the vocabularies below. The mutation-type component of the selected action is additionally injected into the LLM prompt as a structural hint ("prefer `<type>` changes"), steering generation toward the chosen edit class. The full tuple is used when the mutation is applied directly at the AST level (the fallback path when no valid LLM completion is produced).

Table 5: Discrete mutation-type vocabulary for the SAC meta-controller (symbolic regression).

Action	Label	Description
0	`add_operator`	Wrap a node with a unary op, or insert a binary op above it
1	`replace_operator`	Replace an existing operator node (arity-preserving)
2	`delete_node`	Remove a unary wrapper, promoting its child
3	`insert_constant`	Multiply a sub-expression by a constant
4	`subtree_rewrite`	Replace the node's subtree with a fresh random tree

The operator vocabulary comprises 5 binary operators (`add`, `sub`, `mul`, `div`, `pow`) and 13 unary functions (`sin`, `cos`, `tan`, `exp`, `log`, `sqrt`, `abs`, `tanh`, `arcsin`, `arccos`, `arctan`, `sinh`, `cosh`); the constant vocabulary contains 9 values $\{-2, -1, -0.5, 0.5, 1, 2, 3, \pi, e\}$, giving an argument-index space of size $\max(18, 9) = 18$. All expressions are built from the three observable inputs: position x, velocity v, and time t. Numeric constants introduced as `params[j]` remain free parameters optimized by BFGS during evaluation. Generated code applies numerical guards (clipped exponents, ϵ-protected division and logarithms) so that every mutated program remains evaluable.

D ALGORITHM

Algorithm 2 Discrete SAC for Program Mutation

Require: GNN encoder Φ, evaluator $\mathcal{E}$, elite population $\mathcal{P}$, prioritized replay buffer $\mathcal{B}$, warm-up W, total steps S

Initialise policy π_ϕ, twin critics $Q_{\theta_1}, Q_{\theta_2}$, targets $Q_{\bar{\theta}_1}, Q_{\bar{\theta}_2}$, temperature $\log \alpha$
for $s = 1, \ldots, S$ **do**
  $\alpha_{\text{prog}} \sim \text{Uniform}(\mathcal{P})$ {parent program, sampled uniformly}
  $z \leftarrow \Phi(\texttt{build_graph}(\alpha_{\text{prog}}))$
  $a \sim \begin{cases} \text{Uniform}(\mathcal{A}) & \text{if } s < W \\ \pi_\phi(\cdot \mid z) & \text{otherwise} \end{cases}$
  $\beta \leftarrow \texttt{mutate}(\alpha_{\text{prog}}, \texttt{decode}(a))$ {AST edit via `MutableSRTree`}
  $R(\beta) \leftarrow \mathcal{E}(\beta)$ {BFGS constant fit + NMSE}
  $z' \leftarrow \Phi(\texttt{build_graph}(\beta))$
  $r \leftarrow \text{clip}\big(R(\beta), r_{\min}, r_{\max}\big) + \lambda_1\, r_{\text{nov}}(z', \mathcal{P}) - \lambda_2\, r_{\text{cx}}(\beta)$
  Store (z, a, r, z') in $\mathcal{B}$
  if $|\mathcal{B}| \geq 32$ **and** $s \geq W$ **then**
    Sample prioritized minibatch; update $Q_{\theta_1}, Q_{\theta_2}, \pi_\phi$, and α
    Soft-update targets $Q_{\bar{\theta}_1}, Q_{\bar{\theta}_2}$ with Polyak coefficient τ
  end if
  $\mathcal{P}.\text{Insert}(\beta, R(\beta), z')$ {keep top-N by score}
end for

E HYPERPARAMETERS

Table 6 collects all parameters introduced in the methodology (§3.2–§3.5), grouped by component. Values marked † are set in the implementation but not stated explicitly in the main text.

F EXTRA RESULTS

F.1 SYMBOLIC REGRESSION

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.

GAEvolve best program

$$\begin{aligned} f(x, t, v;\, \boldsymbol{\theta}) = {} & \theta_0\, x + \theta_1\, x^2\, \text{sgn}(x) + \theta_2\, x^3 + \theta_3\, x^4\, \text{sgn}(x) \\ & + \theta_4\, v + \theta_5\, v^2\, \text{sgn}(v) + \theta_6\, v^3 \\ & + \theta_7\, \sin(t) + \theta_8\, \cos(t) \\ & + \theta_9\, \big(|x||v| + x^2|v| + |t||v| + t^2 v + x v^2\big) \end{aligned} \tag{10}$$

The terms $\theta_0 x$, $\theta_1 x^2\, \text{sgn}(x)$, $\theta_2 x^3$, and $\theta_3 x^4\, \text{sgn}(x)$ correspond to the linear restoring force, quadratic and cubic nonlinearity, and quartic nonlinearity respectively. The terms $\theta_4 v$, $\theta_5 v^2\, \text{sgn}(v)$, 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 parameter θ_9, are interaction terms.

Table 6: Hyperparameters of GAEvolve, grouped by component.

Parameter	Meaning	Value
Elite archive and evolution loop (§3.2, Alg. 1)		
N	Archive size (top-N elite population)	50
G	Children generated per generation (group size)	8
K	GNN fine-tuning interval (generations)	3
d	Structural embedding dimension	128
GNN encoder (§3.3)		
F	Node-type vocabulary size (one-hot input)	17
$\lvert\mathcal{E}\rvert$	Number of typed edge relations	6
L	Relational message-passing layers	2
h	Hidden width (pooled projection gives $2h = d$)	64
—	Gradient steps per fine-tuning round	300
—	Sliding-window size of $\mathcal{L}$ (recent pairs)	300
—	Minimum samples before fine-tuning ($\lvert\mathcal{L}\rvert$)	10
Discrete SAC meta-controller (§3.4)		
$\lvert\mathcal{A}\rvert$	Action-space size ($5 \times 30 \times 18$)	2700
—	Policy/critic MLP hidden width	256
τ	Polyak target-update coefficient	0.005
W	Warm-up steps (uniform actions)	32
—	Minimum buffer size before updates	32
—	Replay buffer capacity (prioritized)	10,000
—	Mini-batch size	64
γ	Discount factor in the TD target	$0.99^{\dagger}$
—	SAC learning rate (Adam, all networks)	$3 \times 10^{-4\,\dagger}$
$\bar{\mathcal{H}}$	Target entropy (auto-tuned α)	$-\frac{1}{2}\log\lvert\mathcal{A}\rvert$
$r_{\min},\ r_{\max}$	Task-score clipping bounds (Eq. 6)	$-5,\ 15$
λ_1	Novelty bonus weight (Eq. 6)	0.1
λ_2	Complexity penalty weight (Eq. 6)	0.05
—	Complexity penalty onset (AST nodes, Eq. 6)	20
GRPO LLM fine-tuning (§3.5)		
ϵ_{clip}	PPO clipping range (Eq. 8)	0.2
β_{kl}	KL coefficient (monitoring only, Eq. 8)	0.1
—	Crash-sentinel reward threshold (dropped)	-10
—	Advantage-std skip threshold	10^{-8}

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, $\theta_{22}x^2t$, $\theta_{23}v^2t$, and $\theta_{24}xvt$ are cross-terms.

OpenEvolve best program (Gemini)

$$\begin{aligned} f(x,t,v;\boldsymbol{\theta}) = \theta_0\,x + \theta_1\,\sin(t) + \theta_2\,v + \theta_3\,x^2v \\ + \theta_4\,x^2 + \theta_5\,v|v| + \theta_6\,xv^2 + \theta_7\,x^3 \\ + \theta_8\,\cos(t) + \theta_9\,v^3 \end{aligned} \tag{11}$$

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 $\theta_6 xv^2$ 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

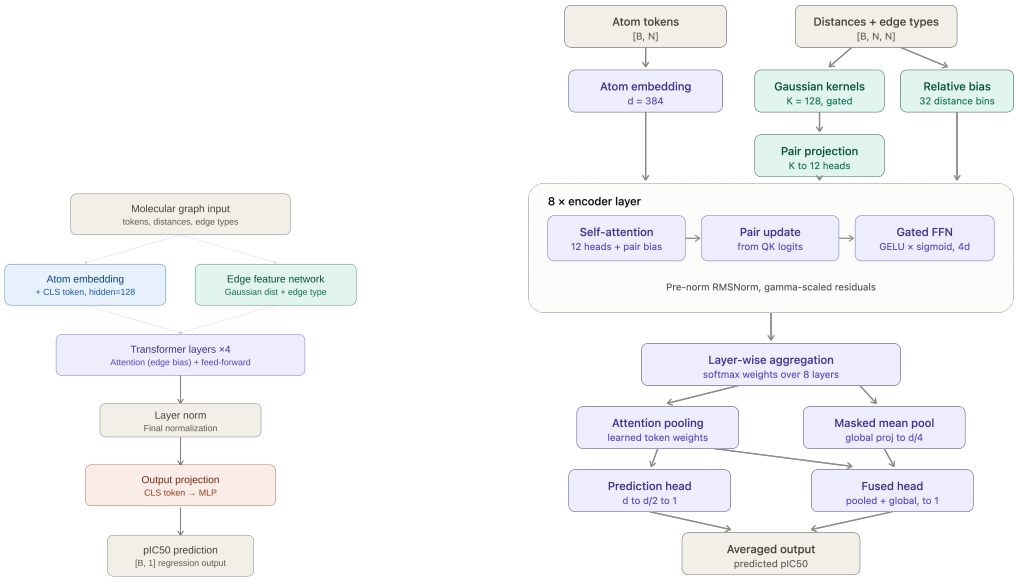
$$\begin{aligned} f(x,t,v;\boldsymbol{\theta}) = \theta_0\,x + \theta_1\,v^3 + \theta_2\,v + \theta_3\,x\,e^{-v^2} + \theta_4\,x^3 \\ + \theta_5\,v\,e^{-v^2} + \theta_6\,xv^2 + \theta_7\,\sin(t) \\ + \theta_8\,\cos(t) + \theta_9\,x|x| \end{aligned} \tag{12}$$

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 $\theta_6 xv^2$ 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.

F.2 Molecular Property Prediction

F.3 PIC50 Prediction

Figure 3 shows the seed architecture and the best architecture found by GAEvolve for pIC50 regression (§4.3). The fitness used during search is $0.7\max(0,R^2) + 0.2\,(1-\min(\mathrm{MAE}/3,1)) + (0.1-\min(n_{\mathrm{param}}/5\times10^6, 0.1))$ on the validation split.



(a) Seed program. (b) Best program found by GAEvolve.

Figure 3: pIC50 architecture search: seed and evolved architectures.