Progressive Subexpression Reuse in Symbolic Regression: Insights from RL-based Search and a Genetic Programming Realization

Xiangdong Wu[1], **Wenjun Wu**[1,2], **Bingrun Chen**[2], **Junle Wang**[1], **Zhaoxin Fan**[1], **Haoyi Zhou**[3], **Pengju Zhang**[4] and **Rongye Shi**[1,2,*]

[1]School of Artificial Intelligence, Beihang University, China
[2]Hangzhou International Innovation Institute, Beihang University, China
[3]School of Software, Beihang University, China
[4]Institute of Automation, Chinese Academy of Sciences, China
{xiangdongwu, wwj09315, chenbr, wjl_admire, zhaoxinf, haoyi, shirongye}@buaa.edu.cn,
pengju.zhang@ia.ac.cn

Abstract

Symbolic regression (SR) aims to recover compact and interpretable mathematical expressions from data. Genetic programming (GP) directly searches over symbolic structures, but its population dynamics can make it difficult to reliably preserve and accumulate useful subexpressions. In contrast, reinforcement learning (RL)-based SR has shown strong empirical performance, suggesting that learned sampling dynamics may capture useful regularities in symbolic search. Motivated by this contrast, we analyze expressions sampled during RL training and identify a recurring pattern, termed *progressive subexpression reuse*, where useful simple subexpressions emerge early, become increasingly frequent, and support the formation of more complex structures. Based on this observation, we propose *Reinforcement Genetic Programming (RGP)*, a purely GP-based and non-RL framework that explicitly realizes a stage-wise retention–reintroduction loop through a dynamically maintained subexpression pool and pool-guided population initialization. Experiments on standard SR benchmarks show that RGP matches or outperforms strong baselines without RL policy training, suggesting that progressive subexpression reuse is an effective mechanism for SR search. We release our code at https://github.com/wuxiangdong586-max/RGP.

1 Introduction

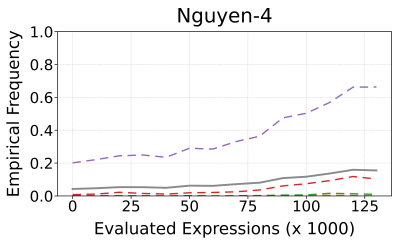
Symbolic regression (SR) aims to infer explicit mathematical expressions from data, producing compact and interpretable models that are especially valuable in scientific discovery [Schmidt and Lipson, 2009; Udrescu and Tegmark, 2020]. In most SR settings, candidate models are represented as symbolic expression trees, and solving SR amounts to searching over a discrete and hierarchical space of such trees. However, the space grows combinatorially with expression size and operator choices, making search inefficient and unstable, particularly on complex problems where many locally plausible structures exist.

Genetic programming (GP) is a classical framework for SR [Koza, 1994], evolving expression trees through selection, mutation, and crossover. Despite decades of progress, GP-based SR can struggle on harder tasks, where success often hinges on discovering useful *building blocks* (subexpressions) and reliably accumulating them so that increasingly complex structures can be composed. Many extensions strengthen the *within-generation* search dynamics by modifying selection and variation, including ϵ-lexicase selection [La Cava *et al.*, 2016], semantic techniques [Virgolin *et al.*, 2019], and linkage-learning-based recombination [Virgolin, 2020].

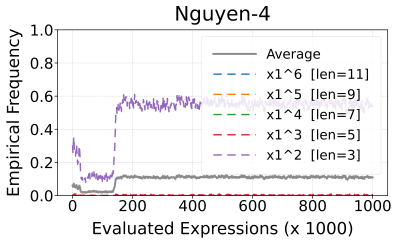
Recently, reinforcement learning (RL) has emerged as a strong alternative for SR [Petersen *et al.*, 2019; Kamienny *et al.*, 2022; Holt *et al.*, 2023]. Beyond final performance, prior analyses suggest that RL-generated solutions often improve progressively during training, with parts of the target expression emerging over time [Du *et al.*, 2024]. Motivated by this, we analyze expression samples during search and track the empirical frequencies of ground-truth subexpressions when the target expression is available. As shown in Fig. 1, RL-based SR (DSR [Petersen *et al.*, 2019]) exhibits a clear hierarchical pattern: simpler subexpressions tend to emerge earlier and become frequent, followed by more complex ones, whereas standard GP shows more oscillatory and less ordered dynamics under the same evaluation budget. Additional results for other RL-based solvers and GP across different tasks are provided in the Appendix.

These observations motivate a mechanism-level hypothesis: effective SR benefits from *progressive subexpression reuse*, a closed-loop process in which promising subexpressions are (i) *retained* from current search outcomes and (ii) *reintroduced* to shape the next-stage sampling/initialization distribution, enabling stable accumulation of increasingly complex structures. This raises a natural question: **can we realize such progressive reuse within GP, without policy training, by intervening only on stage-wise population construction?**

*Corresponding author: Rongye Shi (shirongye@buaa.edu.cn).



(a) DSR



(b) GP

Figure 1: Per-checkpoint frequencies of ground-truth subexpressions for DSR (a) and standard GP (b) on Nguyen-4 ($x^6 + x^5 + x^4 + x^3 + x^2 + x$). Each curve corresponds to a target subexpression grouped by preorder traversal length, with frequencies measured over newly evaluated expressions at budget-aligned checkpoints. The x-axis ranges differ because DSR recovers the target and stops after about 127k evaluations, whereas GP fails to recover it within the 10^6 evaluation budget.

We answer this question affirmatively and propose *Reinforcement Genetic Programming* (RGP), a fully gradient-free SR framework that keeps the GP inner loop unchanged and instead implements progressive reuse at the *process level*. RGP maintains a reusable subexpression pool that is refreshed from each stage's search outcomes and directly guides next-stage population construction via pool-guided initialization. In doing so, RGP offers a simple, training-free instantiation of the retention and reintroduction mechanism suggested by RL dynamics, and yields competitive performance against both GP baselines and learning-guided SR methods under matched evaluation budgets.

Our contributions are as follows:

- We propose a simple, reproducible protocol to track subexpression frequency trajectories during SR search, providing process-level evidence of a typically hierarchical emergence from simpler to more complex components in RL-based SR relative to GP baselines.
- We propose RGP, a process-level retention and reintroduction method that externalizes reusable subexpressions into a pool refreshed from current-stage outcomes and uses it to bias next-stage population construction, while keeping standard GP operators unchanged.
- We demonstrate on standard SR benchmarks that, under matched operator/constraint settings and budget-aligned evaluation, RGP matches or outperforms strong baselines, highlighting the practical value of explicit process-level retention–reintroduction.

2 Related Work

Symbolic regression has been studied from multiple perspectives that differ in how symbolic expressions are generated and optimized. We focus on two lines most relevant to our work: GP-based symbolic regression and learning-guided symbolic regression. Other paradigms, including physics-inspired decomposition, neural symbolic regression, pretrained sequence models, and tree search, have also been explored [Udrescu and Tegmark, 2020; Kim *et al.*, 2020; Kamienny *et al.*, 2022; Holt *et al.*, 2023; Sun *et al.*, 2022].

2.1 Genetic Programming for Symbolic Regression

Symbolic regression has traditionally been studied within the framework of genetic programming (GP), following its original formulation by Koza [Koza, 1994]. In GP-based approaches, candidate expressions are represented as syntax trees and evolved through selection and variation operators, making GP naturally suited to the discrete and hierarchical structure of symbolic expressions.

A longstanding challenge in GP-based symbolic regression is the identification and reuse of *building blocks*, i.e., subexpressions that contribute positively to overall performance. Most existing approaches address this issue implicitly. Pareto-based evolutionary strategies, such as Age–Fitness Pareto optimization (AFP) [Schmidt and Lipson, 2010], introduce auxiliary objectives to promote diversity and mitigate premature convergence, which can indirectly increase the survival of useful subexpressions. Semantic methods further exploit behavioral information of programs. For example, ϵ-lexicase selection [La Cava *et al.*, 2016] favors individuals that perform well on difficult subsets of training cases, while semantic backpropagation techniques [Virgolin *et al.*, 2019] guide recombination based on desired intermediate outputs, encouraging the preservation of semantically meaningful subexpressions. Dependency-aware recombination methods, such as GP-GOMEA [Virgolin, 2020], explicitly model interactions between program components to better preserve cooperating substructures during crossover. Similar reusable-component ideas also appear beyond GP-based SR, especially in program synthesis systems that learn libraries of reusable programs or abstractions, such as DreamCoder [Ellis *et al.*, 2021] and InnateCoder [Moraes *et al.*, 2025].

2.2 Learning-guided Symbolic Regression

Reinforcement learning (RL) has emerged as a strong paradigm for symbolic regression by casting expression construction as a sequential decision-making problem. In this formulation, an agent generates an expression token by token according to a policy and receives a reward based on the data-fitting quality of the resulting expression. A representative example is Deep Symbolic Regression (DSR) [Petersen *et al.*, 2019], which learns a sampling policy over symbolic token sequences via policy optimization, enabling effective exploration in large discrete spaces.

Beyond purely RL-driven generators, a growing line of work combines learned generation with evolutionary or search-based refinement. A common strategy is to use a

learned generator or RL-guided search agent to shape the *starting distribution* of candidate expressions, and then apply GP-style variation and selection for downstream refinement. For example, GEGL [Ahn *et al.*, 2020] uses a learned generator to seed the GP population, while NGGP [Mundhenk *et al.*, 2021] couples evolutionary search with policy learning to bias generation toward high-reward expressions. RSRM [Xu and Sun, 2024] further leverages a reinforcement-guided search agent to seed and steer symbolic refinement. These methods highlight the value of learned sampling or decision biases in symbolic search, while evolutionary operators provide complementary refinement and diversity.

2.3 Position of RGP

RGP is closest to GP reuse mechanisms, linkage-learning GP, and neural/RL-guided population seeding, but it introduces the reuse bias at a different control point. Standard elitism or Hall-of-Fame mechanisms retain complete high-fitness expressions as candidate solutions, whereas RGP extracts their subexpressions and turns them into explicit reusable initialization units. Linkage-learning methods such as GP-GOMEA use dependency information among tree components to guide recombination inside the GP loop; RGP does not learn linkage structure and does not modify selection, crossover, or mutation. Instead, it keeps the GP search loop standard and applies the retained subexpressions only before each stage, by biasing the construction of the next initial population. Neural- or RL-guided seeding methods learn a generator or policy to propose expressions, while RGP derives its bias directly from the evolving Hall-of-Fame-mined subexpression pool. Thus, RGP is not a new genetic operator or a learned proposal model, but an outer-loop retention–reintroduction mechanism for stage-wise population construction.

3 Preliminaries

3.1 Symbolic Trees

In symbolic regression, a mathematical expression is commonly represented as a *symbolic tree*. A symbolic tree is an ordered tree whose internal nodes correspond to operators (e.g., $+$, $\times$, $\sin$), while leaf nodes represent variables or constants; each symbolic tree uniquely defines a mathematical expression, and vice versa.

3.2 Genetic Programming for Symbolic Regression

Genetic programming (GP) is a tree-based evolutionary search paradigm for symbolic regression. It maintains a population of symbolic expression trees initialized either randomly or from predefined primitives. At each generation, new candidate expressions are produced through subtree crossover and mutation, while selection is guided by data-fitting performance. To preserve high-quality solutions, GP-based methods often employ elitism mechanisms, such as a Hall-of-Fame, which retains top-performing expressions across generations.

3.3 Reinforcement Learning–based Symbolic Regression

Reinforcement learning–based symbolic regression formulates expression construction as a sequential decision-making process by generating a symbolic expression as a preorder traversal sequence $\mathbf{s}$, enabling batch generation and optimization of expressions. Given a parameterized policy π_θ, the training objective is to maximize the expected reward over sampled sequences:

$$\max_{\theta} \; \mathbb{E}_{\mathbf{s}\sim\pi_\theta}\left[R(\mathbf{s})\right], \tag{1}$$

where $R(\mathbf{s})$ is defined based on the data-fitting quality of the symbolic expression $\tau(\mathbf{s})$ decoded from sequence $\mathbf{s}$, such as the negative prediction error on the training data.

4 Method

This section presents Reinforcement Genetic Programming (RGP), a fully gradient-free symbolic regression framework that operationalizes the progressive subexpression reuse pattern observed in RL-based SR. RGP implements this idea via a stage-level retention–reintroduction loop: at each stage, we refresh a subexpression pool and use it to bias the initialization of the next population. Importantly, the GP inner loop remains unchanged—selection, crossover, mutation, and constant optimization are all standard—and our only intervention is stage-wise population construction, providing a minimal and attribution-friendly mechanism for reuse without policy training. We refresh the pool between stages (rather than every generation) so that recombination can proceed on a fixed initialization distribution within a stage, yielding more stable genetic search before the memory signal is updated.

4.1 The Framework of RGP

The core idea of RGP is to externalize stage-wise search feedback as reusable subexpressions mined from elite expressions and to reuse them to shape the next-stage initialization distribution, as illustrated in Fig. 2 and summarized in Algorithm 1. We use *stage* to denote the outer closed-loop iteration in Algorithm 1, while *generation* refers to the inner GP evolution within a stage.

RGP consists of three components that operate in a closed loop:

- **Subexpression Pool.** An explicit repository that stores reusable subexpressions mined from elite solutions and is refreshed across stages to balance exploitation and exploration.
- **Pool-guided Population Construction.** A procedure that constructs the initial GP population by incorporating subexpressions sampled from the pool into randomly generated expression trees.
- **Genetic Programming Search.** A standard GP process that evolves the constructed population to produce high-quality symbolic expressions.

At stage t, we (i) refresh the subexpression pool from the hall-of-fame elites obtained up to stage $t-1$, and (ii) use the refreshed pool to construct a pool-guided initial population via injection. A standard GP run then evolves this population, and the newly found elites are fed back to refresh the pool for stage $t+1$. Through repeated stages, this closed-loop process progressively biases GP toward high-potential structures while retaining exploration.

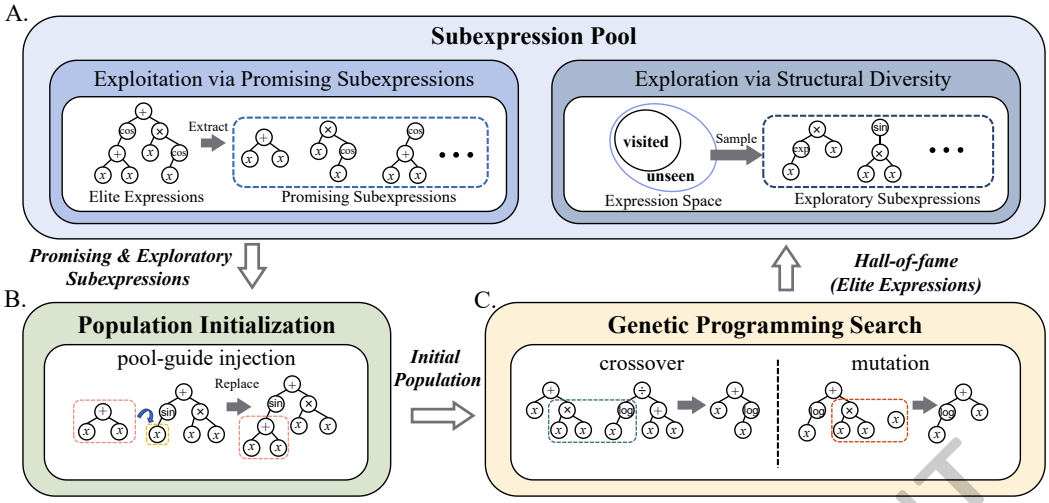


Figure 2: Overview of the proposed Reinforcement Genetic Programming (RGP) framework, which operates in a closed-loop manner. **(A)** Promising subexpressions are extracted from elite expressions discovered so far and stored in a subexpression pool, balancing exploitation of reusable structures and exploration of structural diversity. **(B)** Subexpressions sampled from the pool guide population construction by replacing parts of randomly generated expression trees, biasing search toward promising candidates. **(C)** A standard GP run evolves the initialized population, and the resulting elites are fed back to refresh the subexpression pool for the next stage.

4.2 Subexpression Pool

Fig. 1 suggests that RL-based SR may implicitly realize *progressive subexpression reuse* through its learned sampling distribution: once useful subexpressions appear in high-reward expressions, they tend to reoccur more frequently in subsequent samples and later support the emergence of more complex structures. To examine this hypothesis, we seek a non-RL instantiation that implements the same *retention–reintroduction* loop within GP.

RGP instantiates progressive reuse via an explicit *subexpression pool* that biases population construction while being refreshed by GP search outcomes. Concretely, at stage t we (i) construct the pool using elites available up to stage $t-1$, and (ii) use the pool to construct the initial population for stage t. The pool merges two sources:

$$\mathcal{P}_t = \mathcal{S}^{\text{util}}_{t-1} \cup \mathcal{S}^{\text{exp}}_{t-1}, \tag{2}$$

where $\mathcal{S}^{\text{util}}_{t-1}$ collects reusable subexpressions mined from elites (exploitation), and $\mathcal{S}^{\text{exp}}_{t-1}$ injects novel subexpressions to expand structural coverage and maintain diversity (exploration).

Exploitation via Promising Subexpressions

The exploitation component explicitly retains subexpressions that repeatedly occur in high-quality solutions. At the beginning of stage t, we update the reusable part of the pool by mining subexpressions from the global hall-of-fame $\mathcal{E}_{t-1}$. Conceptually, it proceeds in three steps: *(i) mine candidates from elites*, *(ii) score each candidate*, and *(iii) select a compact reusable set*.

Step 1: Mining candidates from elites. We maintain a global hall-of-fame set $\mathcal{E}_t$ that stores the best top-K expressions found up to stage t, together with their fitness values. Each element $\tau \in \mathcal{E}_t$ is a symbolic expression tree with fitness $r(\tau)$ (e.g., negative MSE), where a larger value indicates better fit. We decompose each $\tau \in \mathcal{E}_{t-1}$ into its subtrees and collect syntactically valid subexpressions. We represent each subexpression s as a subtree, and define its size $|s|$ as the number of tokens (nodes), implemented as the length of its preorder traversal sequence. To avoid overly trivial fragments and overly large structures, we keep only subexpressions with sizes $|s| \in [l_a, h_a]$. Let $\mathcal{S}_b$ denote the deduplicated candidate set:

$$\mathcal{S}_b = \bigcup_{\tau \in \mathcal{E}_{t-1}} \text{Sub}(\tau; [l_a, h_a]), \tag{3}$$

where $\text{Sub}(\tau; [l_a, h_a])$ returns the set of subexpressions (subtrees) of τ whose sizes $|s|$ lie in $[l_a, h_a]$.

Step 2: Scoring candidates by quality and recurrence. Not all candidates in $\mathcal{S}_b$ are equally useful. We summarize each $s \in \mathcal{S}_b$ by (i) its *average fitness* among elites that contain it and (ii) its *occurrence frequency* within the hall-of-fame:

$$\bar{r}(s) = \frac{\sum_{\tau \in \mathcal{E}_{t-1}} r(\tau)\, \mathbb{I}[s \preceq \tau]}{\sum_{\tau \in \mathcal{E}_{t-1}} \mathbb{I}[s \preceq \tau]}, \tag{4}$$

$$p(s) = \frac{1}{|\mathcal{E}_{t-1}|} \sum_{\tau \in \mathcal{E}_{t-1}} \mathbb{I}[s \preceq \tau], \tag{5}$$

where $s \preceq \tau$ indicates that s is a subtree of τ, and $\mathbb{I}[\cdot]$ is the indicator function.

Step 3: Selecting a reusable exploitation set. To form a compact exploitation memory, we rank candidates with a simple hierarchical rule: *longer-first, then higher quality, then higher recurrence*. We prioritize longer subexpressions because they often subsume shorter ones as subtrees and thus act as more informative, compositional building blocks. Moreover, longer subexpressions occur less frequently in elites, so their quality estimates can be noisier; a quality-first rule may prematurely discard potentially useful longer structures. We

Algorithm 1 RGP (overall loop)

Require: Dataset $\mathcal{D}$; operator/grammar $\mathcal{F}$; stages T; population size N; injection prob P_{in}; early-stop threshold ϵ; pool hyperparameters; hall-of-fame size K.
Ensure: Best expression $\tau^\star$.

Notation: $\mathcal{E}_t$ = global hall-of-fame (best-so-far expressions); $\mathcal{H}_t$ = history of used subexpressions; $\mathcal{P}_t$ = subexpression pool.
Initialize $\mathcal{E}_0 \leftarrow \emptyset,\ \mathcal{H}_0 \leftarrow \emptyset,\ \tau^\star \leftarrow \emptyset$
for $t = 1$ to T **do**
  $\mathcal{S}_{t-1}^{\text{util}} \leftarrow \text{UpdateUtil}(\mathcal{E}_{t-1})$ // mine & select from HOF (Sec. 4.2)
  $\mathcal{S}_{t-1}^{\text{exp}} \leftarrow \text{UpdateExp}(\mathcal{H}_{t-1})$ // sample unseen valid (Sec. 4.2)
  $\mathcal{P}_t \leftarrow \mathcal{S}_{t-1}^{\text{util}} \cup \mathcal{S}_{t-1}^{\text{exp}}; \quad \mathcal{H}_t \leftarrow \mathcal{H}_{t-1} \cup \mathcal{P}_t$
  Sample base population $\{\tau_i^{(0)}\}_{i=1}^N \sim \text{RandInit}(\mathcal{F})$
  $\{\tau_i^{\text{init}}\}_{i=1}^N \leftarrow \text{Inject}(\{\tau_i^{(0)}\}, \mathcal{P}_t, P_{\text{in}})$ // subtree replacement (Sec. 4.3)
  $\hat{\tau}_t \leftarrow \text{GP}(\{\tau_i^{\text{init}}\}_{i=1}^N, \mathcal{D})$ // best-of-stage (standard GP)
  $\mathcal{E}_t \leftarrow \text{SelectTop}(\mathcal{E}_{t-1} \cup \{\hat{\tau}_t\};\ K,\ r)$
  $\tau^\star \leftarrow \arg\max_{\tau \in \mathcal{E}_t \cup \{\tau^\star\}} r(\tau)$
  if $\text{MSE}(\tau^\star; \mathcal{D}) \le \epsilon$ **then**
    break
  end if
end for
return $\tau^\star$

therefore use fitness and recurrence only to break ties among similarly sized candidates.

Concretely, we sort $s \in \mathcal{S}_b$ by decreasing $|s|$, then decreasing $\bar{r}(s)$, and finally decreasing $p(s)$. To reduce redundancy, we prune any candidate that is a strict subtree of a previously retained one under the same ranking. Finally, we keep the top-k candidates, denoted as

$$\mathcal{S}_{t-1}^{\text{util}} \;=\; \text{Select}(\mathcal{S}_b;\ |s|\downarrow,\ \bar{r}(s)\downarrow,\ p(s)\downarrow), \tag{6}$$

where $\downarrow$ denotes descending-order sorting. This set serves as the exploitation set for population construction at stage t.

Exploration via Structural Diversity
To mitigate premature convergence, RGP includes an explicit exploration component that promotes structural diversity in the subexpression pool. We maintain a lightweight history set $\mathcal{H}_t$ to record previously used subexpressions. Exploration candidates are drawn from the set of *valid* subexpressions, i.e., syntactically valid trees under the operator/grammar constraints with sizes $|s| \in [l_b, h_b]$. Since this range is bounded, the candidate set is finite; in practice, we choose a small h_b and pre-enumerate all valid candidates with $|s| \in [l_b, h_b]$ once before search into a cache.

At the end of stage $t-1$, we sample exploration subexpressions while excluding any candidate already in $\mathcal{H}_{t-1}$:

$$\mathcal{S}_{t-1}^{\text{exp}} \;\subseteq\; \{\, s :\ |s| \in [l_b, h_b],\ s \notin \mathcal{H}_{t-1} \,\}. \tag{7}$$

After forming the pool at stage t, we record all subexpressions used in the pool:

$$\mathcal{H}_t \;\leftarrow\; \mathcal{H}_{t-1} \cup \mathcal{P}_t. \tag{8}$$

This history-based sampling encourages exploration of unseen structures and helps maintain diversity in the pool.

4.3 Population Construction Guided by the Subexpression Pool

Given a subexpression pool $\mathcal{P}_t$, the remaining question is how to use this search feedback to improve symbolic search.

Recent hybrid symbolic regression methods [Mundhenk *et al.*, 2021] often rely on a trained neural generator to propose candidate expressions as an initial population, followed by GP for downstream refinement. By shaping the *starting distribution* of candidates, such approaches can work synergistically with GP's evolutionary operators and improve the efficiency of subsequent search.

Following this insight, we use the subexpression pool to guide GP population construction at each stage. Specifically, at stage t, we first generate a base population $\{\tau_i^{(0)}\}_{i=1}^N$ using standard random tree initialization. We then perform pool-guided injection with a fixed probability P_{in} per individual. For each base individual $\tau_i^{(0)}$, we sample a Bernoulli gate $z_i \sim \text{Bernoulli}(P_{\text{in}})$. If $z_i = 1$, we sample a small set (a fixed number in all experiments; see Appendix for details) of subexpressions $\mathcal{S}_{\text{in}}^{(i)} \subset \mathcal{P}_t$ and apply a subtree-replacement operator that replaces randomly selected subtrees of $\tau_i^{(0)}$ with subexpressions from $\mathcal{S}_{\text{in}}^{(i)}$ to obtain a candidate $\tilde{\tau}_i$; otherwise we keep the original individual. We accept the replacement only if the resulting tree satisfies standard GP validity constraints (syntactic consistency and a maximum size limit). If a sampled replacement is invalid, we resample locations/subexpressions up to a small fixed number of attempts; otherwise we keep the original tree. Formally,

$$\tau_i^{\text{init}} = \begin{cases} \tilde{\tau}_i, & z_i = 1 \ \wedge\ \text{Valid}(\tilde{\tau}_i), \\ \tau_i^{(0)}, & \text{otherwise.} \end{cases} \tag{9}$$

The constructed population $\{\tau_i^{\text{init}}\}_{i=1}^N$ is then evolved by a standard GP run at stage t.

To further mitigate stagnation, RGP employs a lightweight restart rule: if the reusable exploitation set remains unchanged for a fixed number of consecutive stages, we clear the subexpression pool (optionally retaining the single best individual) and reinitialize the process, encouraging the search to escape local optima.

5 Experiments

In this section, we conduct comprehensive experiments to evaluate RGP on standard symbolic regression benchmarks, including comparative studies, ablation analyses, and efficiency evaluation.

5.1 Evaluation Metrics

We evaluate performance using the recovery rate [Petersen *et al.*, 2019], defined as the fraction of independent runs that exactly recover the ground-truth expression. Compared to evaluations based on fixed datasets [Xu and Sun, 2024] or data-fitting metrics such as MSE [Jiang *et al.*, 2024], this metric

provides a more faithful assessment of symbolic reconstruction capability. Formally, the recovery rate is defined as:

$$\text{Recovery Rate} = \frac{1}{J}\sum_{j=1}^{J}\mathbb{I}\left[\mathcal{A}(D_j) \equiv f\right], \tag{10}$$

where D_j denotes a dataset randomly sampled from the ground-truth function f, $\mathcal{A}$ is the symbolic regression algorithm, J is the number of independent runs, and $\equiv$ indicates symbolic equivalence between expressions. Unless stated otherwise, we compare methods under a matched evaluation budget measured by the total number of fitness evaluations (i.e., evaluated candidate expressions).

5.2 Baselines and Benchmarks

In our comparison, we include the following baseline methods in symbolic regression: **GP**, the classical genetic programming algorithm for symbolic regression; **DSR**, a policy gradient–based method that formulates expression generation as a sequential decision process [Petersen *et al.*, 2019]; **NGGP**, which employs an RNN to initialize the GP population and iteratively updates the RNN using GP search results [Mundhenk *et al.*, 2021]; **DGSR**, a transformer-based pretrained generative model for symbolic regression [Holt *et al.*, 2023]; **RSRM**, which combines double Q-learning with Monte Carlo Tree Search to balance exploration and exploitation [Xu and Sun, 2024]; and **UDFS**, which performs unbiased symbolic regression by systematically searching the space of expression DAGs [Kahlmeyer *et al.*, 2025].

For fair comparison, we use the same operator set and expression constraints (e.g., size/depth limits) across methods whenever applicable; method-specific settings follow the original implementations.

To assess the stability and efficiency of our approach, we evaluate it on three widely used symbolic regression benchmarks. These include: **Nguyen**, a widely used benchmark of symbolic expressions with one or two independent variables; **R***, which consists of three rational functions with polynomial terms in both numerator and denominator, designed to increase structural complexity; and **Livermore**, a challenging set of equations involving high-degree exponentials, trigonometric components, and highly nonlinear polynomials.

5.3 Main Results

As shown in Table 1, under our matched operator/constraint settings and evaluation budgets, RGP achieves the highest (or tied-highest) recovery rates among the evaluated methods across the considered benchmarks. These improvements are consistent with our hypothesis that stage-wise retention and reintroduction of promising subexpressions can stabilize the accumulation of building blocks during GP search. RGP makes this mechanism explicit by mining reusable subexpressions from a global Hall-of-Fame of elites into a pool, which then guides population construction across stages, providing a more persistent reuse signal than relying only on within-generation survival.

5.4 SRBench Results

We further evaluate RGP on SRBENCH [La Cava *et al.*, 2021], which contains both white-box and black-box symbolic regression tasks. Specifically, we consider (i) the SRBENCH Feynman Equations suite (100 physics formulas with available ground-truth expressions) and (ii) the standard SRBENCH black-box tasks where the underlying functional forms are unknown. Throughout, we follow the official SRBENCH protocol, including the prescribed train/test splits and metric definitions. For the Feynman suite, we report the *solution rate* under the benchmark definition, i.e., the fraction of equations for which the recovered expression exactly matches the target up to symbolic equivalence, and we additionally apply the benchmark's additive Gaussian noise injection to the *training* targets under multiple relative noise levels to emulate measurement noise. For black-box tasks, where exact recovery is not defined, we report the SRBENCH test metric as predictive performance together with the corresponding model size (number of nodes), reflecting the accuracy–simplicity tradeoff. On the Feynman data, RGP achieves superior solution rates compared to the baselines, while on the black-box tasks, it strikes a balance between predictive accuracy and model simplicity, achieving competitive R^2 performance with relatively simpler expressions. Additional implementation details are provided in the Appendix.

5.5 Ablation

We conduct ablation studies on the Livermore benchmark to isolate the contributions of key components in our framework. Table 2 reports symbolic recovery rates averaged over multiple independent runs.

Model 1 (Full) corresponds to the complete framework, which maintains a subexpression pool with both exploitation (reusing promising subexpressions) and exploration (using structurally novel components), together with pool-guided subtree replacement during population initialization and standard GP search. **Model 2 (w/o Exploitation)** disables the reuse of promising subexpressions mined from elites, while retaining exploration and the replacement operation, isolating the role of exploitation in enabling progressive reuse. **Model 3 (w/o Exploration)** restricts the pool to promising subexpressions only, removing the use of novel components to assess how much diversity contributes beyond reuse. **Model 4 (Random Pool)** preserves the overall framework but replaces the pool contents with randomly sampled subexpressions matched in size, testing whether gains arise from meaningful reuse rather than the replacement operation itself. **Model 5 (Pure GP)** removes the subexpression pool entirely, reducing the method to standard GP with random restarts.

Overall, **Model 3 (w/o Exploration)** remains competitive with **Model 1 (Full)**, whereas variants that disable **Exploitation** degrade substantially, indicating that retaining and reintroducing promising subexpressions is particularly important for RGP's performance.

5.6 Time Efficiency

We study the time–accuracy tradeoff of RGP by varying the *initialization budget* B_{init}, defined as the number of candidate expressions constructed for stage-wise population initialization (i.e., before GP evolution), while keeping the remaining GP inner-loop computation fixed (e.g., population size, number of generations, and fitness evaluations per stage). For

Bench. (N)	GP	DSR	DGSR	NGGP	RSRM	UDFS	RGP
Nguyen (13)	56.92 ± 28.49	77.69 ± 23.42	74.23 ± 25.33	85.77 ± 21.07	86.54 ± 17.93	69.62 ± 26.53	95.38 ± 9.19
Livermore (22)	17.50 ± 11.28	31.14 ± 16.97	77.27 ± 15.61	79.09 ± 15.60	77.95 ± 15.09	60.91 ± 21.20	87.05 ± 13.44
R* (3)	0.00 ± 0.00	0.00 ± 0.00	25.00 ± 56.92	13.33 ± 57.37	88.33 ± 25.86	33.33 ± 143.42	93.33 ± 28.68
Overall (38)	$\mathbf{29.61 \pm 12.77}$	$\mathbf{44.61 \pm 14.60}$	$\mathbf{72.11 \pm 12.73}$	$\mathbf{76.18 \pm 12.66}$	$\mathbf{81.71 \pm 10.23}$	$\mathbf{61.71 \pm 15.39}$	$\mathbf{90.39 \pm 8.18}$

Table 1: Recovery rate (%) of several algorithms on the Nguyen, Livermore, and R* benchmark problem sets. For each task, the recovery rate is computed over 20 independent runs. Results are averaged across all tasks in each benchmark, with uncertainties reported as 95% confidence intervals of the benchmark-level mean, computed from task-level recovery rates using Student's t interval.

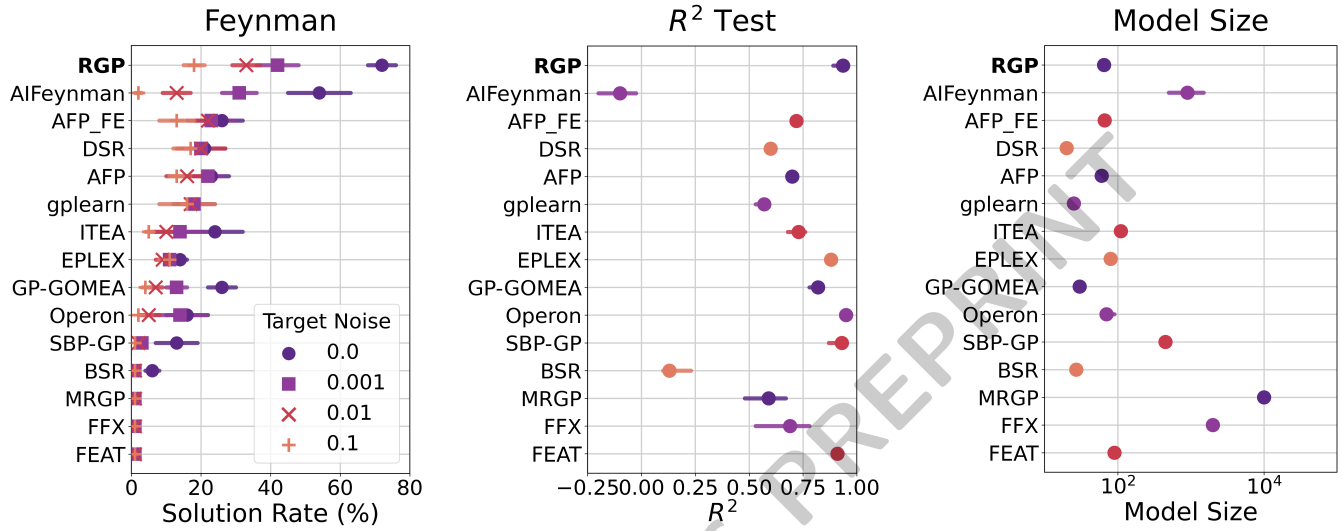


Figure 3: Experimental results on SRBENCH. Left: Feynman solution rate under different training-noise levels (per SRBENCH). Middle/Right: black-box results reporting the SRBENCH test metric (predictive performance) and the corresponding model size (number of nodes).

Method	Mean (%)	Std (%)
FULL	87.27	30.42
w/o Exploitation	33.64	39.10
w/o Exploration	83.64	32.00
Random pool	31.36	40.27
pure GP	30.00	38.79

Table 2: Ablation study on the Livermore tasks. Recovery rates (%) are averaged over 10 independent runs.

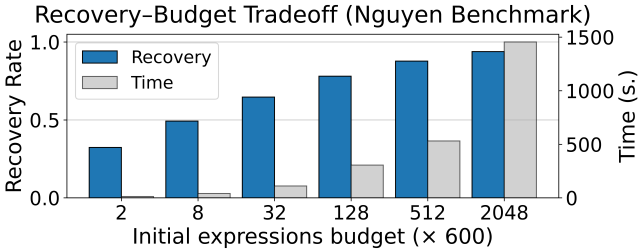


Figure 4: Tradeoff Experiments

each budget level, we run RGP on the full Nguyen benchmark and report the recovery rate averaged across all Nguyen tasks together with the corresponding wall-clock time measured under a fixed hardware/software setup (Appendix). As shown in Fig. 4, increasing B_{init} consistently improves recovery but exhibits diminishing returns at larger budgets, whereas runtime grows monotonically, quantifying a practical knob to trade compute for symbolic recovery.

6 Conclusion

We analyzed the training dynamics of RL-based symbolic regression and found empirical evidence consistent with *progressive subexpression reuse*, where useful subexpressions emerge early, become increasingly sampled, and later support the construction of more complex structures. Motivated by this mechanism-level perspective, we proposed *Reinforcement Genetic Programming* (RGP), a purely GP-based approach that explicitly retains and reintroduces promising subexpressions via a dynamically maintained pool, without relying on RL. Experiments on standard benchmarks show that RGP achieves competitive or superior performance with significantly reduced reliance on expensive policy training. These results suggest that modeling how useful subexpressions are retained and reused over search can be a major factor in effective symbolic regression.

Acknowledgments

This work was supported by the National Science and Technology Major Project (2022ZD0117801), the National Natural Science Foundation of China (62306023), and the Beijing Nova Program (20240484490).

References

[Ahn *et al.*, 2020] Sungsoo Ahn, Junsu Kim, Hankook Lee, and Jinwoo Shin. Guiding deep molecular optimization with genetic exploration. *Advances in neural information processing systems*, 33:12008–12021, 2020.

[Du *et al.*, 2024] Mengge Du, Yuntian Chen, and Dongxiao Zhang. Discover: Deep identification of symbolically concise open-form partial differential equations via enhanced reinforcement learning. *Physical Review Research*, 6(1):013182, 2024.

[Ellis *et al.*, 2021] Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Lucas Morales, Luke Hewitt, Luc Cary, Armando Solar-Lezama, and Joshua B Tenenbaum. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd acm sigplan international conference on programming language design and implementation*, pages 835–850, 2021.

[Holt *et al.*, 2023] Samuel Holt, Zhaozhi Qian, and Mihaela van der Schaar. Deep generative symbolic regression. In *The Eleventh International Conference on Learning Representations (ICLR 2023)*, 2023.

[Jiang *et al.*, 2024] Nan Jiang, Md Nasim, and Yexiang Xue. Vertical symbolic regression via deep policy gradient. In *Proceedings of the Thirty-Third International Joint Conference on Artificial Intelligence*, pages 5891–5899, 2024.

[Kahlmeyer *et al.*, 2025] Paul Kahlmeyer, Joachim Giesen, Michael Habeck, and Henrik Voigt. Scaling up unbiased search-based symbolic regression. *arXiv preprint arXiv:2506.19626*, 2025.

[Kamienny *et al.*, 2022] Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers. *Advances in Neural Information Processing Systems*, 35:10269–10281, 2022.

[Kim *et al.*, 2020] Samuel Kim, Peter Y Lu, Srijon Mukherjee, Michael Gilbert, Li Jing, Vladimir Čeperić, and Marin Soljačić. Integration of neural network-based symbolic regression in deep learning for scientific discovery. *IEEE transactions on neural networks and learning systems*, 32(9):4166–4177, 2020.

[Koza, 1994] John R Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and computing*, 4(2):87–112, 1994.

[La Cava *et al.*, 2016] William La Cava, Lee Spector, and Kourosh Danai. Epsilon-lexicase selection for regression. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 741–748, 2016.

[La Cava *et al.*, 2021] William La Cava, Bogdan Burlacu, Marco Virgolin, Michael Kommenda, Patryk Orzechowski, Fabrício Olivetti de França, Ying Jin, and Jason H Moore. Contemporary symbolic regression methods and their relative performance. *Advances in neural information processing systems*, 2021(DB1):1, 2021.

[Moraes *et al.*, 2025] Rubens O Moraes, Quazi Asif Sadmine, Hendrik Baier, and Levi HS Lelis. Innatecoder: learning programmatic options with foundation models. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, pages 7652–7660, 2025.

[Mundhenk *et al.*, 2021] T Nathan Mundhenk, Mikel Landajuela, Ruben Glatt, Claudio P Santiago, Daniel M Faissol, and Brenden K Petersen. Symbolic regression via neural-guided genetic programming population seeding. *arXiv preprint arXiv:2111.00053*, 2021.

[Petersen *et al.*, 2019] Brenden K Petersen, Mikel Landajuela, T Nathan Mundhenk, Claudio P Santiago, Soo K Kim, and Joanne T Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. *arXiv preprint arXiv:1912.04871*, 2019.

[Schmidt and Lipson, 2009] Michael Schmidt and Hod Lipson. Distilling free-form natural laws from experimental data. *science*, 324(5923):81–85, 2009.

[Schmidt and Lipson, 2010] Michael D Schmidt and Hod Lipson. Age-fitness pareto optimization. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 543–544, 2010.

[Sun *et al.*, 2022] Fangzheng Sun, Yang Liu, Jian-Xun Wang, and Hao Sun. Symbolic physics learner: Discovering governing equations via monte carlo tree search. *arXiv preprint arXiv:2205.13134*, 2022.

[Udrescu and Tegmark, 2020] Silviu-Marian Udrescu and Max Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science advances*, 6(16):eaay2631, 2020.

[Virgolin *et al.*, 2019] Marco Virgolin, Tanja Alderliesten, and Peter AN Bosman. Linear scaling with and within semantic backpropagation-based genetic programming for symbolic regression. In *Proceedings of the genetic and evolutionary computation conference*, pages 1084–1092, 2019.

[Virgolin, 2020] Marco Virgolin. *Design and Application of Gene-pool Optimal Mixing Evolutionary Algorithms for Genetic Programming.* PhD thesis, Delft University of Technology, Netherlands, 2020.

[Xu and Sun, 2024] Yilong Xu and Hao Sun. Reinforcement symbolic regression machine. In *The Twelfth International Conference on Learning Representations*, 2024.