LEARNING TO DESIGN SR SOLVERS: HARNESS OPTIMIZATION FOR SYMBOLIC REGRESSION

Anonymous authors
Paper under double-blind review

ABSTRACT

Symbolic regression (SR) seeks explicit equations that explain observed data, and recent methods increasingly employ large language models (LLMs) and agents for equation discovery. However, these systems typically rely on manually designed workflows and prompts. We argue that the SR agent harness should itself be an optimization target: its design shapes search priors and how numerical feedback guides subsequent equation refinement. We therefore propose **HarnessSR**, a framework for jointly optimizing prompts and workflows to construct reusable SR agents. During both harness optimization and agent execution, HarnessSR supports either PySR or an LLM-based equation generator through tool-specific interfaces, allowing the framework to optimize search guidance for different equation-discovery mechanisms. HarnessSR first optimizes role prompts through equation-discovery feedback in basic workflows, then jointly searches prompt assignments and workflows with brief adaptation to align the two. Finally, it refines role instructions through reflection on residual patterns, candidate equation structures, and search outcomes, improving how these signals guide subsequent equation search. On LSR-Synth in LLM-SRBench, HarnessSR achieves a mean ID $\mathrm{Acc}_{0.01}$ of 61.33% across three backbones, a 45.0% relative improvement over the next-best baseline.

1 INTRODUCTION

Symbolic regression (SR) seeks explicit mathematical relationships that explain observed data (Wang et al., 2023; Zhang et al., 2025b; Karniadakis et al., 2021; Krenn et al., 2022; Udrescu & Tegmark, 2020; Udrescu et al., 2020). The resulting equations can be inspected, interpreted, and manipulated, and related to conservation laws, dimensional constraints, and domain theories. SR thus goes beyond numerical prediction to turn observations into testable scientific hypotheses, supporting equation discovery across physics, biology, materials science, and other domains.

Recent advances in large language models (LLMs) and agents have introduced a new paradigm for SR. These methods combine the scientific knowledge and reasoning capabilities of LLMs with numerical search, evaluation, and feedback to iteratively propose, test, and revise equation hypotheses (Xia et al., 2025; Shojaee et al., 2025a; Yang et al., 2026; Zhao et al., 2026). Central to these systems is the *solver harness*: the executable layer of role instructions, tool interfaces, and control logic that governs interactions between the LLM, data, and numerical tools. The harness governs how scientific knowledge informs candidate proposals and how numerical feedback guides decisions to retain, revise, or replace them. These decisions shape whether subsequent computation explores alternative functional forms, addresses structural errors, or repeats attempts, thereby affecting the quality of equations recovered within a given budget.

However, recent SR solver largely focus on designing solver harnesses by hand, rather than treating the solver harness itself as an object of systematic optimization. Recent research shows that, even with the base model fixed, harness choices can affect both task performance and inference resource use (Lee et al., 2026; Zhou et al., 2026). Solver harness optimization extends beyond adding new SR roles. Within a fixed workflow, different instructions and few-shot demonstrations can shift the model's output distribution. Conversely, with SR roles and prompts held fixed, changes in workflow topology—including invocation frequency and parallelism—can alter information flow, computational allocation, and final outcomes. These choices influence how data analysis informs equation

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.

search, how fitting and residual feedback guide candidate revision, and how finite computation is allocated between exploration and refinement. **The SR agent harness is therefore not merely an execution scaffold, but a design variable to be optimized for equation discovery.** This motivates us to develop a method for automatically optimizing the harness for reusable SR agent, aiming to improve both the quality and efficiency of equation discovery.

To this end, we introduce **HarnessSR**, a framework for optimizing reusable SR solver harnesses. We formulate harness design as the joint optimization of role instructions, demonstrations, and workflow organization to obtain an SR agent harness capable of discovering higher-quality equations. To address the strong coupling among these components and tailor their behavior to SR, we adopt a three-stage procedure. First, we optimize role prompts within basic workflows to establish strong starting points for joint search. Second, we jointly optimize workflows and prompt assignments, with brief adaptation to accommodate their interdependence. Third, we use a reflection model to diagnose failures in SR execution traces and refine role instructions, enabling the harness to translate residual patterns, candidate equation structures, and search outcomes into targeted search guidance. This guidance is applied through tool-specific interfaces that translate agent-generated search plans into parameters and inputs for two selectable base SR tools (we integrates PySR and an LLM-based equation generator as agent tools). These interfaces allow agents to steer the chosen tool during both harness optimization and deployment, while keeping its underlying search algorithm fixed. The top-performing harness is then frozen and reused across new SR tasks, with model weights and the underlying search algorithms held fixed. Across three backbones on LSR-Synth, HarnessSR achieves a mean ID $\mathrm{Acc}_{0.01}$ of 61.33%, a 45.0% relative improvement over the next-best baseline.

Our main contributions are: (1) We highlight that the solver agent harness can also be regarded as an optimization target for SR task. Its design shapes the LLM's reasoning process, influencing the priors that guide equation search, how search feedback is interpreted and used, and ultimately the quality of the discovered equations. (2) We propose HarnessSR, an SR agent optimization framework that jointly optimizes prompts and workflows and uses reflection on SR execution traces to adapt harness behavior to equation discovery. (3) We demonstrate a 45.0% relative improvement in mean ID $\mathrm{Acc}_{0.01}$ over the next-best baseline across three backbones on LSR-Synth.

2 Related Work

Symbolic Regression. Classical SR methods include genetic programming, which evolves expressions through crossover, and mutation (Koza, 1994; Stephens, 2024; Pawlak & Krawiec, 2018; Moraglio et al., 2012; de Franca & Aldeia, 2021). PySR combines evolutionary search with expression simplification and constant optimization (Cranmer, 2023). Neural-guided methods such as DSR learn a distribution over expressions using a recurrent neural network and guide candidate generation through risk-seeking policy gradients (Petersen et al., 2021; Zhang et al., 2025a; Li et al.; Meidani et al., 2023; Kamienny et al., 2022; Li et al., 2022; Shojaee et al., 2023; Yu et al., 2025).

LLM-based methods leverage the generative capacity, prior knowledge, and reasoning capabilities of LLMs to guide equation search. LLM-SR and SGA couple LLM-generated structural hypotheses with numerical parameter optimization (Shojaee et al., 2025a; Ma et al., 2024). Beyond candidate generation, LaSR introduces a learned concept library to guide evolutionary search (Grayeli et al., 2024), while RESTART uses residual feedback to guide an LLM in revising candidate equation structures (Li & Pan, 2026). Deliberate Evolution guides LLM-based equation generation with adaptive search operators, structural diagnostics, and reflective memory (Pang et al., 2026).

Agentic methods extend this paradigm through tool use and feedback-driven interaction. SR-Scientist autonomously invokes data-analysis and equation-evaluation tools to refine hypotheses (Xia et al., 2025). KeplerAgent extracts physical properties, including symmetries, to configure numerical SR engines Yang et al. (2026). A-SR introduces hierarchical coordination to adapt role selection, memory use, and feedback routing during search (Zhao et al., 2026).

Despite increasingly autonomous execution, the overarching harness in representative SR systems remains largely hand-designed. For example, SR-Scientist selects tools autonomously, but does so within predefined tool interfaces, instruction templates, and outer-loop control rules. Such autonomy adapts the search within a given harness without optimizing the harness itself. Yet these design choices govern how hypotheses are proposed and how evidence guides subsequent search, mak-

ing them integral to the equation-discovery strategy. Therefore, we focus on optimizing the solver harness under a limited budget to obtain stronger, reusable SR solvers.

Self-Evolving Agents. Self-evolution and automated agent design extend optimization from individual outputs to the procedures that produce them. MIPRO optimizes instructions and demonstrations using end-to-end task feedback (Opsahl-Ong et al., 2024). MASS optimizes prompts for individual building blocks, selects how these blocks are composed, and then refines their prompts within the selected workflow (Zhou et al., 2026), while Meta-Harness searches executable code surrounding a fixed language model using execution feedback (Lee et al., 2026). However, existing methods have devoted limited attention to harness optimization for automated scientific discovery. In our paper, we show that optimizing the solver harness can substantially improve both the quality and efficiency of scientific discovery.

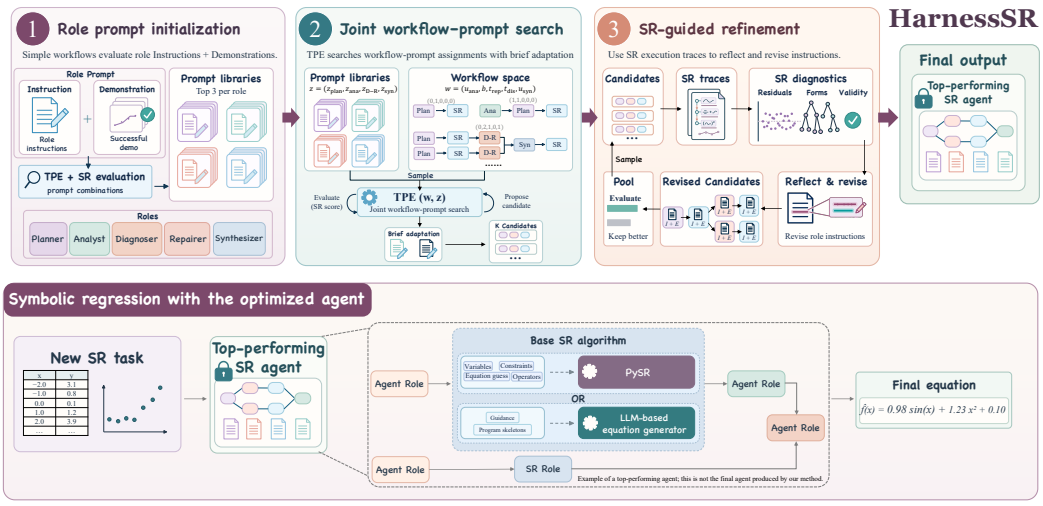

Figure 1: Overview of HarnessSR. It initializes role prompts, jointly optimizes workflows and prompt assignments, and refines instructions with SR-specific feedback. The top-performing agent harness is then frozen and reused on new tasks. The agent workflow shown below is illustrative.

3 METHOD

Unlike prior SR methods that manually design a new solver, we optimize the solver harness itself. For an SR agent, HarnessSR jointly searches role prompts, activation, parallelism, and iteration. To coordinate these coupled components for equation discovery, we adopt a three-stage procedure. The resulting top-performing SR agent is reusable and better coordinates scientific reasoning with numerical search under limited compute.

3.1 PROBLEM FORMULATION

Given an SR task q, typically comprising scientific context, variable descriptions, and observed data, an SR agent S seeks an explicit equation explaining the observations. With the base language model M fixed, S is instantiated from its harness $\mathcal{H}$ and returns $\hat{f}_q$:

$$S = \text{Instantiate}(M, \mathcal{H}), \qquad \hat{f}_q = S(q).$$

We aim to optimize SR agents by optimizing their harnesses without changing the LLM weights.

We define $\mathcal{H} = (\mathcal{R}, w, \mathcal{F})$, where $\mathcal{R}$ specifies the typical roles in SR agent and their prompts:

$$\mathcal{R} = \{(r, P_r) \mid r \in \{\text{Planner}, \text{Analyst}, \text{Diagnoser}, \text{Repairer}, \text{Synthesizer}\}\}.$$

These roles respectively construct equation search plans, extract evidence of observation, diagnose equation deficiencies, revise plans based on diagnoses, and integrate findings across branches. The w controls the workflow structure. $\mathcal{F}$ contains fixed tools and components shared by all candidates,

including a base SR tool such as PySR (Cranmer, 2023) and an LLM-based equation generator, candidate management, and the final equation selector.

For each role r, the full prompt includes runtime context, such as task information, search history, and feedback from other roles, together with two optimizable components $P_r = (I_r, E_r)$: an instruction I_r and a demonstration set E_r. The instruction I_r specifies the role's objectives, reasoning requirements, and action principles. The demonstration set E_r illustrates the role's task through input–output examples and may be empty. These components can alter the LLM's reasoning strategies and outputs even with roles and workflows fixed.

We define the workflow using five discrete variables:

$$w = (u_{\mathrm{ana}}, b, t_{\mathrm{rep}}, t_{\mathrm{dis}}, u_{\mathrm{syn}}) .$$

These variables jointly control role activation, parallel search, and iterative refinement, as summarized in Tab. 1.

Table 1: Effects of workflow parameters. PySR as an example of base SR tools.

Parameter	Values	Effect on the workflow
u_{ana}	$\{0, 1\}$	Activates *Analyst* to examines observational evidence before initial planning.
b	$\{1, 2, 3, 4, 5\}$	Number of parallel search branches, each executing *Planner* $\rightarrow$ PySR, or *Planner* $\rightarrow \cdots \rightarrow$ PySR
t_{rep}	$\{0, 1, 2, 3\}$	Number of search-plan refinement rounds in each branch, following *Diagnoser* $\rightarrow$ *Repairer* $\rightarrow$ PySR.
t_{dis}	$\{0, 1, 2\}$	Number of cross-branch discussion rounds; in each round, each branch uses the shared results to update its own search plan and run PySR.
u_{syn}	$\{0, 1\}$	Activates *Synthesizer* to integrate branch findings into a unified plan, followed by one additional PySR run.

The task-initiating *Planner* is always active and needs no control parameter. The *Diagnoser–Repairer* pair is active when $t_{\mathrm{rep}} > 0$. Single-branch workflows omit cross-branch replanning and synthesis, requiring $t_{\mathrm{dis}} = u_{\mathrm{syn}} = 0$ when $b = 1$. Sec. 3.2 details the execution order and information flow.

We jointly optimize role instructions, demonstrations, and workflow organization on a training set $\mathcal{D}$ to maximize average equation quality under workflow-validity and execution-budget constraints:

$$(\mathcal{R}^\star, w^\star) \in \underset{\substack{\mathcal{R},\, w \\ w \in \mathcal{W}_B}}{\arg\max}\, J_{\mathcal{D}}(\mathcal{H}), \text{ where } J_{\mathcal{D}}(\mathcal{H}) = \frac{1}{|\mathcal{D}|} \sum_{q \in \mathcal{D}} Q(S(q), q) .$$

Here, $\mathcal{W}_B$ contains workflows satisfying role dependencies and a per-task execution budget B. Each candidate evaluation executes the complete solver and scores its final equation using the fixed quality score Q. The stage-specific optimization sets $\mathcal{D} = \cup\{\mathcal{D}_i\}_{i=1}^{3}$ and selection set $\mathcal{D}_{\mathrm{eva}}$ are mutually disjoint random subsets of the public training set released by Xia et al. (2025). Each $\mathcal{D}_i$ is used for optimization in Stage i, while $\mathcal{D}_{\mathrm{eva}}$ is used for candidate selection during harness optimization. The three-stage optimization of HarnessSR approximately solves this objective to obtain the top-performing SR solver agent, and then applies it to new SR tasks.

3.2 Execution within an SR Agent Harness

Given task q, the harness optionally invokes the *Analyst* to examine observations, then initializes b branches. Each branch executes a *Planner*-generated plan using a base SR tool, followed by t_{rep} *Diagnoser–Repairer* refinement rounds. In each of the t_{dis} cross-branch replanning rounds, each *Planner* uses a shared snapshot of preceding results to initiate another search. An optional *Synthesizer* produces one additional plan. Each role shares its prompt P_r across branches and rounds, while runtime context changes.

We integrate two selectable base SR algorithms as agent tools: PySR or an LLM-based equation generator, through a fixed, tool-specific adapter. For PySR, role outputs specify variable subsets, operator sets (`unary_operators`, `binary_operators`), structural restrictions (`constraints`, `nested_constraints`), and expression seeds (`guesses`) to constrain and seed evolutionary search. For the LLM-based generator, the adapter incorporates search guidance and structural hypotheses into the generation prompt and converts expression guesses into initial parameterized program skeletons. The generator retains its iterative equation-program generation, numerical parameter fitting, and evaluation procedures. For both tools, harness updates guide search through tool inputs while leaving the underlying algorithms unchanged.

Harness updates modify the parameters passed to the base SR algorithm. The shared components $\mathcal{F}$ collect valid candidates and select the best equation $\hat{f}_q$, which is scored by an external evaluator as Q. Subsequent stages optimize prompts and workflows through this execution pipeline.

3.3 STAGE 1: ROLE-LEVEL PROMPT OPTIMIZATION

Stage 1 needs to identify a strong base prompt set for each single role, providing a starting point for the joint optimization of prompts and workflow in Stage 2. We optimize instructions I_r and demonstrations E_r of each role r within simple workflows, retaining three high-scoring prompts per role for subsequent search and adaptation.

Optimization of Planner' base prompts. We first fix the base workflow $w = (0, 1, 0, 0, 0)$, which structure is detailed in Tab. 2, and optimize the *Planner*'s instructions I_{Planner} and demonstrations E_{Planner}. By running the solver agent on $\mathcal{D}_1$, we extract input–output demonstrations from successful solver trajectories and use them, together with task information, to generate candidate instructions. Tree-structured Parzen estimator (TPE) (Bergstra et al., 2011) searches over instruction–demonstration combinations, evaluating each through complete solver agent executions on $\mathcal{D}_{\text{eva}}$. We retain the three highest-scoring combinations as the Planner's base prompt packages for subsequent workflow search.

Table 2: Local workflows for prompt initialization. PySR as an example of base SR tools.

Target	Initialization workflow	w
Planner	Planner → PySR → Selection	(0,1,0,0,0)
Analyst	Analyst → Planner → PySR → Selection	(1,1,0,0,0)
Diag–Re	Planner → PySR → Diagnoser → Repairer → PySR → Selection	(0,1,1,0,0)
Synthesizer	Two parallel [Planner → PySR] branches → Synthesizer → PySR → Selection	(0,2,0,0,1)

Optimization of other roles' base prompts. With the *Planner*'s base prompt fixed, we optimize each remaining role within its corresponding workflow in Tab. 2, using the same prompt optimization procedure. Only the target roles' instructions and demonstrations are updated; all other components remain fixed. *Diagnoser* and *Repairer* prompts are optimized and retained as paired configurations.

Prompt libraries. We remove duplicate and invalid versions, and retain the three highest-scoring prompts per role. Diagnoser and Repairer prompts remain paired, yielding four libraries:

$$\mathcal{L}_r = \{P_r^{(1)}, P_r^{(2)}, P_r^{(3)}\}, \quad r \in \{\text{Planner}, \text{Analyst}, \text{DiagRepair}, \text{Synthesizer}\}.$$

Each package contains one $P_r = (I_r, E_r)$, or a paired diagnosis–repair configuration.

These prompts are initial candidates rather than final configurations. Stage 2 jointly optimizes both workflow and prompts, while Stage 3 further refines the instructions using SR execution feedback for task adaptation.

3.4 STAGE 2: JOINT WORKFLOW AND PROMPT OPTIMIZATION

Stage 2 jointly searches workflows and prompt assignments, then briefly adapts the selected prompts to their workflows before choosing candidates for Stage 3.

Candidate agent initialization. We initialize each agent candidate by sampling a workflow and then assigning prompts to it.

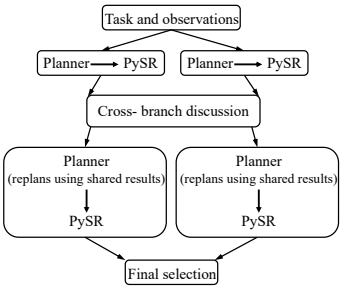

Figure 2: 2-branch search with 1 cross-branch discussion round: w=(0,2,0,1,0). PySR as an example.

Firstly, we construct workflow space $\mathcal{W}_B$ by enumerating the parameter combinations in Tab. 1 and removing those that violate role dependencies or execution-budget constraints. We select n distinct workflows from $\mathcal{W}_B$, which include six basic workflows (the four listed in Tab. 2, two-branch independent search $(0, 2, 0, 0, 0)$, and cross-branch replanning $(0, 2, 0, 1, 0)$, illustrated in Fig. 2). The remaining workflows are sampled uniformly without replacement from $\mathcal{W}_B$.

Secondly, for each sampled workflow, we assign a prompt to every role by probabilistic sampling from the corresponding Stage 1 library: higher-scoring packages are more likely to be selected, while lower-scoring ones remain available for exploration. We repeat this assignment twice for each workflow, resampling duplicate assignments, to obtain $2n$ initial harnesses.

We represent each candidate agent by (w, z), where w specifies the workflow and $z = (z_{\text{Planner}}, z_{\text{Analyst}}, z_{\text{DiagRepair}}, z_{\text{Synthesizer}})$ specifies its prompt assignment. Each $z_r \in \{1, 2, 3\}$ selects package $P_r^{(z_r)}$ from $\mathcal{L}_r$, determining the corresponding role prompts P_r in $\mathcal{R}$. Only indices for active role groups are used; *Diagnoser* and *Repairer* share one package index.

Joint search with TPE. We use the TPE (Bergstra et al., 2011) to optimize (w, z): the five workflow parameters in w and up to four active prompt-package indices in z. We first evaluate the $2n$ initial harnesses on $\mathcal{D}_2$ and initialize TPE with their configurations and scores. At each iteration, TPE divides evaluated configurations into high-scoring candidates and the remainder, models their parameter distributions, and proposes a new (w, z) favored by the high-scoring distribution relative to the other. The resulting solver is evaluated on the same panel, and its score updates the search history. This loop can change both workflow structure and prompt assignment across the full valid space. Therefore, during TPE search, the prompts obtained in Stage 1 remain fixed and are only reassigned across different workflows. Invalid or previously evaluated configurations are excluded.

Brief prompt adaptation for top candidates. TPE can only search for better prompt–workflow combinations within a fixed space and cannot generate new prompts. Because prompts and workflows are tightly coupled, Stage 1 prompts may become suboptimal for the workflows found in Stage 2 and bias their evaluation. We therefore briefly refine the each role's instruction I_r of promising candidates before allocating the larger Stage 3 optimization budget.

We select the best K candidate agents with different workflows based on joint search and fix their workflows and demonstrations. For every role of each top candidate, we ask an LLM to propose a small set of instruction variants informed by role input, output, and execution feedback. TPE then searches for combinations of these variants, evaluating each combination through complete solver executions on a small subset of $\mathcal{D}_2$. Finally, we retain the best instruction combination for each top candidate, effectively fine-tuning its prompts.

3.5 Stage 3: SR-Guided Instruction Refinement

Brief adaptation improves prompt–workflow compatibility but cannot fully address persistent failures in equation search. Stage 3 therefore refines the K Stage 2 candidates using SR-specific execution feedback. For each candidate, we fix the workflow w, demonstrations E_r, and shared components $\mathcal{F}$, and optimize only the role instructions I_r to better adapt them to SR tasks.

SR diagnostic feedback. Each workflow maintains a separate candidate pool $\mathcal{C}_w$, initialized with its adapted harness from Stage 2. At each iteration, we uniformly sample a parent $\mathcal{H}_w$ from the pool and execute it on $\mathcal{D}_3$, recording every role's input and output, search plans, candidate equations, and observational residuals.

An external diagnostic LLM examines these traces and reports identified problems, supporting evidence, and affected roles. It checks and constructs four types of diagnostic feedback: (i) evidence-to-action alignment, checking whether residual patterns and diagnoses lead to concrete search-plan changes; (ii) structural exploration, examining whether branches and rounds explore distinct equation forms and improve candidate quality; (iii) constraint justification, identifying variable, operator, or structural restrictions unsupported by available evidence; and (iv) expression quality, examining fit, complexity, and numerical validity to inform subsequent planning.

This step diagnoses behavior without rewriting instructions or accessing reference equations, ensuring that the diagnostics are not tailored to specific training problems.

Feedback-guided instruction revision. The reflection model (Agrawal et al., 2026) receives the active roles' current instructions and the diagnostic feedback, proposes evidence-supported changes, and produces revised instructions for every roles of candidate workflow. The resulting prompts form a child harness $\mathcal{H}'_w = (\mathcal{R}' = \{(r, P'_r)\}, w, \mathcal{F})$.

Determining the Final SR Harness. Parent and child are evaluated on $\mathcal{D}_{\text{eva}}$. The child enters $\mathcal{C}_w$ only if

$$J_{\mathcal{D}_{\text{eva}}}(\mathcal{H}') > J_{\mathcal{D}_{\text{eva}}}(\mathcal{H}).$$

Otherwise, it is discarded. We repeat this process until the optimization budget is exhausted.

We evaluate the candidates in each pool on $\mathcal{D}_3$ and retain the highest-scoring harness for each of the K workflows. We then evaluate these K harnesses on $\mathcal{D}_3 \cup \mathcal{D}_{\text{eva}}$ and select the top-performing one as the final SR agent harness.

4 EXPERIMENTS

We evaluate HarnessSR's accuracy and efficiency, compare harness optimization with model-weight post-training, and analyze the contributions of optimization stages, workflow search, and diagnostic feedback. The appendices provide experimental configurations and role instructions (C–E and G), alongside optimization costs (A), symbolic accuracy (B), alternative SR backend comparisons (E), out-of-distribution results (F), equation-recovery cases (H), and training-set-size analysis (I).

Table 3: ID accuracy (%) and mean solving time (min/task) on LSR-Synth. Bold indicates the best result in each column.

Method	Overall		Physics		Chemistry		Biology		Materials		Time ↓
	$\mathrm{Acc}_{0.01}$	$\mathrm{Acc}_{0.001}$	$\mathrm{Acc}_{0.01}$	$\mathrm{Acc}_{0.001}$	$\mathrm{Acc}_{0.01}$	$\mathrm{Acc}_{0.001}$	$\mathrm{Acc}_{0.01}$	$\mathrm{Acc}_{0.001}$	$\mathrm{Acc}_{0.01}$	$\mathrm{Acc}_{0.001}$	min/task
Non-LLM methods											
GPLearn	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.15
E2E	0.78	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.00	0.00	0.43
NeSymReS	4.65	1.29	0.00	0.00	4.63	0.00	5.56	0.00	12.00	6.67	15.84
DSR	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.66
uDSR	24.55	10.08	6.82	5.30	43.52	20.37	23.61	6.94	29.33	6.67	1.52
PySR	32.82	16.54	25.00	18.94	24.07	12.96	16.67	9.72	74.67	24.00	**0.21**
Qwen3.6-35B-A3B											
LaSR	10.08	4.65	6.82	4.55	8.33	0.00	4.17	4.17	24.00	12.00	25.61
LLM-SR	34.11	10.85	34.09	6.82	27.78	2.78	12.50	4.17	64.00	36.00	31.15
SR-Scientist	46.51	27.13	31.82	13.64	33.33	13.89	54.17	37.50	84.00	60.00	65.35
HarnessSR	**65.37**	**38.76**	**53.03**	32.58	**61.11**	22.22	**68.06**	**41.67**	**90.67**	**70.67**	3.66
Qwen3-Coder-30B-A3B-Instruct											
LaSR	8.79	6.46	4.55	3.79	12.04	9.26	5.56	5.56	14.67	8.00	9.60
LLM-SR	29.72	9.04	26.52	7.58	37.04	12.96	19.44	6.94	34.67	8.00	23.23
SR-Scientist	32.30	16.02	18.18	8.33	22.22	5.56	22.22	8.33	81.33	52.00	34.72
HarnessSR	57.36	37.73	37.12	33.33	58.33	**28.70**	61.11	27.78	88.00	68.00	4.03
GPT-OSS-20B											
LaSR	16.28	10.08	12.88	7.58	20.37	14.81	13.89	8.33	18.67	9.33	8.24
LLM-SR	28.94	10.85	18.18	3.79	24.07	4.63	18.06	2.78	65.33	40.00	39.58
SR-Scientist	48.06	25.32	35.61	16.67	54.63	25.93	40.28	22.22	68.00	42.67	15.37
HarnessSR	61.24	35.66	47.73	**36.36**	58.33	19.44	66.67	**41.67**	84.00	52.00	3.29

4.1 EXPERIMENTAL SETUP

Data. Following SR-Scientist (Xia et al., 2025), we evaluate on LSR-Synth, the synthetic subset of LLM-SRBench (Shojaee et al., 2025b), comprising 129 tasks across physics, chemistry, biology, and materials science. Harness optimization uses the 1,024 public synthetic training tasks released by SR-Scientist. The effect of training-set size is examined in Appendix I. Dataset details are provided in Appendix C.1.

Models and baselines. We use Qwen3.6-35B-A3B, Qwen3-Coder-30B-A3B-Instruct, and GPT-OSS-20B. Qwen3.6-35B-A3B supports the main results and ablations; Qwen3-Coder-30B-A3B-Instruct supports the comparison with model-weight post-training; and GPT-OSS-20B tests applicability to another model family with fewer total parameters. The non-LLM baselines are GPLearn, E2E, NeSymReS, DSR, uDSR, and PySR. The LLM-based baselines are LaSR, LLM-SR, and SR-Scientist; SR-Scientist+RL is used for the post-training comparison. Model and search settings are given in Appendices C and E; baseline settings are in Appendix D.

Metrics. Following LLM-SRBench (Shojaee et al., 2025b) and SR-Scientist (Xia et al., 2025), we use accuracy-to-tolerance, Acc_τ. For each task, we discard the 5% of test points with the largest relative errors and check whether all remaining errors are at most τ. We report the proportion of tasks meeting this criterion at $\tau = 0.01$ and 0.001, together with mean solving time per task. Evaluation and timing details are provided in Appendices C.2 and C.3. We measure optimization effort by the total number of solver executions. Each execution is one run of an SR solver on one task and may include multiple LLM and SR-tool calls; repeated runs on the same task are counted separately. Symbolic-equivalence accuracy (SA) and supplementary OOD results are reported in Appendices B and F, respectively.

4.2 Main results

Table 3 shows that HarnessSR's mean ID $\mathrm{Acc}_{0.01}$ and $\mathrm{Acc}_{0.001}$ across the three backbones are 45.0% and 63.8% higher than those of the next-best baseline, SR-Scientist, respectively. Mean solving time is reduced by 58.0–85.7% relative to the fastest LLM-based baseline in each model group. Joint prompt–workflow optimization and SR-guided instruction refinement help translate solving feedback into more effective equation search.

These gains extend across scientific domains. HarnessSR leads at both accuracy thresholds in all four domains with the two Qwen models, and in physics, biology, and materials science with GPT-OSS-20B.

4.3 Comparison with reinforcement-learning post-training

Table 4: Two optimization routes based on Qwen3-Coder-30B-A3B-Instruct and their ID accuracy (%).

Method	Optimized component	ID $\mathrm{Acc}_{0.01}$ ↑	ID $\mathrm{Acc}_{0.001}$ ↑
SR-Scientist	None	32.30	16.02
SR-Scientist+RL	Model weights	40.92	20.69
HarnessSR	Prompts and workflow	**57.36**	**37.73**

Table 4 compares weight post-training and harness optimization using the same Qwen3-Coder base model. HarnessSR achieves 57.36% ID $\mathrm{Acc}_{0.01}$, compared with 40.92% for SR-Scientist+RL, and also leads at ID $\mathrm{Acc}_{0.001}$. HarnessSR keeps model weights fixed and improves guidance for SR tools by optimizing role search instructions and their interactions. This result supports harness optimization as an effective route to improving symbolic regression performance. The one-time investments for both optimization routes are given in Appendix A.

4.4 LLM calls versus accuracy

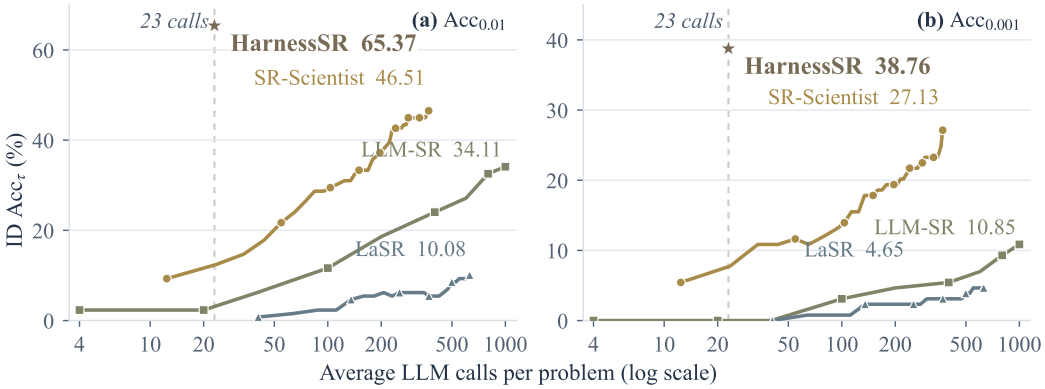

Figure 3: Mean LLM calls per task versus ID accuracy on LSR-Synth (logarithmic horizontal axis). Baseline curves show call checkpoints; HarnessSR shows the full execution of its selected solver configuration. Curve construction is detailed in Appendix D.1.

Figure 3 compares accuracy and LLM calls using Qwen3.6-35B-A3B. HarnessSR achieves the highest accuracy at both thresholds with an average of 23 calls per task, about 6.2% of the call count at SR-Scientist's final point. SR-Scientist, LLM-SR, and LaSR improve as calls accumulate, but their final accuracies remain below HarnessSR despite hundreds or up to a thousand calls. One-time optimization determines role search instructions, branch organization, and interaction rounds; the resulting configuration is then used directly on new tasks. The figure shows that reusing this configuration achieves higher accuracy with fewer model calls.

Table 5: ID accuracy (%) and cumulative solver executions after each optimization stage (Qwen3.6-35B-A3B, LSR-Synth). The initial harness uses initial prompts and a single-branch workflow.

Stage	ID $\mathrm{Acc}_{0.01}$ ↑	ID $\mathrm{Acc}_{0.001}$ ↑	Solver executions
Initial harness	5.43	1.55	0
After Stage 1	27.91	16.54	1,792
After Stage 2	43.41	32.82	2,752
After Stage 3 (full)	**65.37**	**38.76**	4,728

4.5 Contribution of optimization stages

Table 5 shows that all three optimization stages improve accuracy. Stage 1 optimizes role prompts within basic workflows, establishing a useful starting point for the full solver. Stage 2 jointly searches workflows and prompt assignments, then briefly adapts role instructions to the selected workflows so that prompts fit their role combinations and interactions. ID $\mathrm{Acc}_{0.001}$ rises from 16.54% to 32.82%.

Stage 3 raises ID $\mathrm{Acc}_{0.01}$ from 43.41% to 65.37% and ID $\mathrm{Acc}_{0.001}$ from 32.82% to 38.76%. SR diagnostics connect candidate equations, residual patterns, and role actions, checking whether solving feedback leads to changes in search plans and whether different branches explore distinct equation structures. The reflection model uses these diagnostics to revise role instructions, improving guidance for subsequent equation search while keeping the workflow fixed.

4.6 Workflow search and diagnostic feedback

Table 6: Workflow search and feedback on LSR-Synth (Qwen3.6-35B-A3B): ID accuracy (%) and mean LLM calls per task. Fixed-pool search selects presampled workflows; full-space search can propose new ones. Instructions are revised using execution traces directly or SR diagnostics of these traces.

Workflow search	Refinement feedback	ID $\mathrm{Acc}_{0.01}$ ↑	ID $\mathrm{Acc}_{0.001}$ ↑	LLM calls/task ↓
Fixed-pool search	Execution traces	49.35	28.17	37
Full-space search	Execution traces	53.49	33.07	**22**
Fixed-pool search	SR diagnostics	56.59	32.30	37
Full-space search	SR diagnostics	**65.37**	**38.76**	23

Table 6 shows that SR diagnostics improve accuracy at both thresholds under both search strategies, with larger gains in full-space search. The diagnostics connect candidate-equation structures and residual patterns with role actions, giving instruction revision concrete directions for equation search. This feedback guides role-prompt optimization more effectively than using execution traces directly, including when more workflows are available. Combining full-space search with SR diagnostics raises ID $\mathrm{Acc}_{0.01}$ from 49.35% to 65.37% and reduces mean LLM calls per task from 37 to 23, compared with fixed-pool search using execution traces.

5 Conclusion and Limitation

We introduced HarnessSR, a framework that treats the SR agent harness itself as an optimization target. It jointly optimizes role instructions, demonstrations, and workflow organization while keeping model weights and underlying search algorithms fixed. Results on LSR-Synth support this approach as a way to construct stronger, reusable SR agents. A central implication for SR is that numerical feedback can inform both equation refinement within a task and the development of search strategies reused across tasks. Through harness optimization, evidence from training tasks shapes how agents propose hypotheses, interpret residuals, and organize exploration and refinement on new problems. Prompts and workflows therefore encode consequential choices about how equation discovery proceeds. This perspective motivates studying solver organization as an optimization dimension in SR, alongside equation representations and search algorithms.

However, the present formulation searches within a predefined workflow space. Whether more open-ended workflow structures can reveal additional reusable strategies for equation discovery remains an important question for future research.

THE USE OF LARGE LANGUAGE MODELS (LLMS)

Within HarnessSR, LLMs perform search planning, observational analysis, equation diagnosis, plan revision, and integration of findings across branches. During harness optimization, they generate candidate role instructions, analyze execution traces, and refine instructions using SR-specific feedback. The framework also supports an LLM-based equation generator as an alternative SR tool. Model weights remain fixed throughout harness optimization and solver execution.

Beyond the use of LLMs in the HarnessSR described above, LLMs served solely as auxiliary tools for polishing and refining the manuscript's language. Their contributions to research conception, experimental design, or manuscript preparation do not warrant authorship or contributor status. All substantive research ideas, methods, and results presented in this work were independently developed or produced by the authors.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In International Conference on Learning Representations, volume 2026, pp. 8479–8565, 2026.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. Advances in neural information processing systems, 24, 2011.

Luca Biggio, Tommaso Bendinelli, Alexander Neitz, Aurelien Lucchi, and Giambattista Parascandolo. Neural symbolic regression that scales. In International Conference on Machine Learning, pp. 936–945. Pmlr, 2021.

Miles Cranmer. Interpretable machine learning for science with pysr and symbolicregression. jl. arXiv preprint arXiv:2305.01582, 2023.

Fabricio Olivetti de Franca and Guilherme Seidyo Imai Aldeia. Interaction–transformation evolutionary algorithm for symbolic regression. Evolutionary computation, 29(3):367–390, 2021.

Arya Grayeli, Atharva Sehgal, Omar Costilla Reyes, Miles Cranmer, and Swarat Chaudhuri. Symbolic regression with a learned concept library. Advances in Neural Information Processing Systems, 37:44678–44709, 2024.

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.

George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. Nature Reviews Physics, 3(6):422–440, 2021.

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

Mario Krenn, Robert Pollice, Si Yue Guo, Matteo Aldeghi, Alba Cervera-Lierta, Pascal Friederich, Gabriel dos Passos Gomes, Florian Häse, Adrian Jinich, AkshatKumar Nigam, et al. On scientific understanding with artificial intelligence. Nature Reviews Physics, 4(12):761–769, 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.

Yoonho Lee, Roshen Nair, Qizheng Zhang, Kangwook Lee, Omar Khattab, and Chelsea Finn. Meta-harness: End-to-end optimization of model harnesses. arXiv preprint arXiv:2603.28052, 2026.

Qian Li, Yuxiao Hu, Juncheng Liu, and Yuntian Chen. Gensr: Symbolic regression based on equation generative space. In The Fourteenth International Conference on Learning Representations.

Wenqiang Li, Weijun Li, Linjun Sun, Min Wu, Lina Yu, Jingyi Liu, Yanjie Li, and Songsong Tian. Transformer-based model for symbolic regression via joint supervised learning. In The Eleventh International Conference on Learning Representations, 2022.

Yunlun Li and Sinno Jialin Pan. Robust equation structure learning with adaptive refinement. In International Conference on Learning Representations, volume 2026, pp. 70029–70053, 2026.

Pingchuan Ma, Tsun-Hsuan Wang, Minghao Guo, Zhiqing Sun, Joshua B Tenenbaum, Daniela Rus, Chuang Gan, and Wojciech Matusik. Llm and simulation as bilevel optimizers: A new paradigm to advance physical scientific discovery. arXiv preprint arXiv:2405.09783, 2024.

Kazem Meidani, Parshin Shojaee, Chandan K Reddy, and Amir Barati Farimani. Snip: Bridging mathematical symbolic and numeric realms with unified pre-training. arXiv preprint arXiv:2310.02227, 2023.

Alberto Moraglio, Krzysztof Krawiec, and Colin G Johnson. Geometric semantic genetic programming. In International Conference on Parallel Problem Solving from Nature, pp. 21–31. Springer, 2012.

Krista Opsahl-Ong, Michael J Ryan, Josh Purtell, David Broman, Christopher Potts, Matei Zaharia, and Omar Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, pp. 9340–9366, 2024.

Xinyu Pang, Zhanke Zhou, Xuan Li, Fangrui Lv, Shanshan Wei, Sen Cui, Bo Han, and Changshui Zhang. Deliberate evolution: Agentic reasoning for sample-efficient symbolic regression with llms. arXiv preprint arXiv:2606.04360, 2026.

Tomasz P Pawlak and Krzysztof Krawiec. Competent geometric semantic genetic programming for symbolic regression and boolean function synthesis. Evolutionary computation, 26(2):177–212, 2018.

Brenden K Petersen, Mikel Landajuela Larma, Terrell N Mundhenk, Claudio Prata Santiago, Soo Kyung Kim, and Joanne Taery Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In International Conference on Learning Representations, 2021.

Parshin Shojaee, Kazem Meidani, Amir Barati Farimani, and Chandan Reddy. Transformer-based planning for symbolic regression. Advances in Neural Information Processing Systems, 36: 45907–45919, 2023.

Parshin Shojaee, Kazem Meidani, Shashank Gupta, Amir Barati Farimani, and Chandan K Reddy. Llm-sr: Scientific equation discovery via programming with large language models. In The Thirteenth International Conference on Learning Representations, 2025a.

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, 2025b.

Trevor Stephens. gplearn: Genetic programming in python, with a scikit-learn inspired api, 2024. URL https://github. com/trevorstephens/gplearn. Accessed, pp. 05–22, 2024.

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

Silviu-Marian Udrescu, Andrew Tan, Jiahai Feng, Orisvaldo Neto, Tailin Wu, and Max Tegmark. Ai feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. Advances in Neural Information Processing Systems, 33:4860–4871, 2020.

Hanchen Wang, Tianfan Fu, Yuanqi Du, Wenhao Gao, Kexin Huang, Ziming Liu, Payal Chandak, Shengchao Liu, Peter Van Katwyk, Andreea Deac, et al. Scientific discovery in the age of artificial intelligence. Nature, 620(7972):47–60, 2023.

Shijie Xia, Yuhan Sun, and Pengfei Liu. Sr-scientist: Scientific equation discovery with agentic ai. arXiv preprint arXiv:2510.11661, 2025.

Jianke Yang, Ohm Venkatachalam, Mohammad Kianezhad, Sharvaree Vadgama, and Rose Yu. Think like a scientist: Physics-guided llm agent for equation discovery. arXiv preprint arXiv:2602.12259, 2026.

Zihan Yu, Jingtao Ding, and Yong Li. Discover network dynamics with neural symbolic regression. Nature Computational Science, pp. 1–13, 2025.

Hengzhe Zhang, Qi Chen, Wolfgang Banzhaf, Mengjie Zhang, et al. Rag-sr: Retrieval-augmented generation for neural symbolic regression. In The Thirteenth International Conference on Learning Representations, 2025a.

Xuan Zhang, Limei Wang, Jacob Helwig, Youzhi Luo, Cong Fu, Yaochen Xie, Meng Liu, Yuchao Lin, Zhao Xu, Keqiang Yan, et al. Artificial intelligence for science in quantum, atomistic, and continuum systems. Foundations and Trends® in Machine Learning, 18(4):385–849, 2025b.

Wenxiao Zhao, Dong Liu, Kaiyi Xu, Feng Liu, Zhen Zhao, Fei Ben, Shu Wang, Wenhao Li, Yingnian Wu, Fenghua Ling, et al. A-sr: Self-evolving agentic llms for symbolic regression via hierarchical coordination. arXiv preprint arXiv:2608.04872, 2026.

Han Zhou, Xingchen Wan, Ruoxi Sun, Hamid Palangi, Shariq Iqbal, Ivan Vulić, Anna Korhonen, and Sercan Arik. Multi-agent design: Optimizing agents with better prompts and topologies. In International Conference on Learning Representations, volume 2026, pp. 15844–15872, 2026.

A One-time Optimization Cost

The main text compares two adaptation routes on the same Qwen3-Coder-30B-A3B-Instruct base model: SR-Scientist+RL updates model weights through reinforcement learning, whereas HarnessSR keeps the weights fixed and searches reusable role prompts and solver workflows. Table 7 reports the one-time investment required to obtain each final solver; the training task counts and solver execution counts refer to the optimization stage.

Table 7: One-time optimization investment for two adaptation routes.

Method	Training tasks	Solver executions	Optimization GPUs	Final product
SR-Scientist+RL	1,024	15,360	32× H200	Updated model weights
HarnessSR	160	4,728	2× H100	Role prompts and solver workflow

Because of computational constraints, we did not reproduce the reinforcement learning training of SR-Scientist. Its evaluation results, training configuration, and model information are taken from the original paper, official code repository, and Hugging Face model page (Xia et al., 2025). Its 15,360 trajectories follow from 60 training steps, 32 prompts per step, and eight samples per prompt. Each sampled task-level trajectory counts as one solver execution in Table 7. We deploy vLLM on two H100 GPUs to serve concurrent model requests during Harness search.

B Symbolic Accuracy

Following the evaluation method of SR-Scientist (Xia et al., 2025), we use GPT-OSS-120B to assess the symbolic equivalence of predicted and ground-truth equations and report symbolic accuracy (SA).

Table 8: Symbolic accuracy (SA, %) on LSR-Synth.

Non-LLM methods		**LLM-based methods**			
Method	**SA ↑**	**Method**	**Qwen3.6**	**Qwen3-Coder**	**GPT-OSS**
uDSR	0.52	LaSR	6.46	2.84	5.68
PySR	5.17	LLM-SR	5.43	4.91	4.65
		SR-Scientist	8.79	6.98	7.24
		HarnessSR	**10.34**	**12.66**	**11.63**

Qwen3.6: Qwen3.6-35B-A3B; Qwen3-Coder: Qwen3-Coder-30B-A3B-Instruct; GPT-OSS: GPT-OSS-20B.

C Data, Evaluation, and Model Settings

C.1 Training Tasks and Scientific Domains

LLM-SRBench (Shojaee et al., 2025b) comprises LSR-Synth and LSR-Transform. LSR-Synth combines terms from established scientific equations with newly constructed terms, whereas LSR-Transform modifies existing equations. Following SR-Scientist (Xia et al., 2025), we use LSR-Synth because transformed equations may retain clues encountered during model pretraining. Each task provides observed data and separate in-distribution (ID) and out-of-distribution (OOD) test points.

Harness optimization uses a pool of 1,024 public synthetic training tasks released with SR-Scientist (Xia et al., 2025). The training-scale comparison uses 80, 160, 320, and 640 tasks. Training tasks provide role demonstrations, assess candidate prompts and workflows, and supply solver feedback for instruction revision. The final solver harness is frozen before evaluation on the 129 official LSR-Synth tasks.

LSR-Synth spans physics, chemistry, biology, and materials science. Table 9 summarizes representative tasks, input variables, and prediction targets in the four domains, following the examples in LLM-SRBench (Shojaee et al., 2025b).

Table 9: Four scientific domains and evaluation task counts in LSR-Synth.

Domain	Typical problem	Observed variables and equation target	Tasks
Physics	Damped oscillators and other dynamics	Time t, position $x(t)$, velocity $v(t)$; acceleration relation	44
Chemistry	Reaction kinetics	Time t, reactant concentration $c(t)$; rate of concentration change	36
Biology	Population growth and dynamics	Time t, population $P(t)$; population growth rate	24
Materials science	Stress–strain relations	Strain ϵ and temperature T; stress σ	25
Total			129

C.2 Locked-Equation Evaluation and Aggregation

Each method outputs one locked equation per task. The external evaluator computes its predictions on separate in-distribution (ID) and out-of-distribution (OOD) test points; the solver receives observed data, variable descriptions, and scientific context. We use accuracy-to-tolerance, Acc_τ, following LLM-SRBench (Shojaee et al., 2025b) and SR-Scientist (Xia et al., 2025). For either distribution, let task q have n_q test points, with target $y_{q,i}$ and locked-equation prediction $\hat{y}_{q,i}$ at point i. For nonzero targets, the relative error is

$$e_{q,i} = \frac{|\hat{y}_{q,i} - y_{q,i}|}{|y_{q,i}|}. \tag{1}$$

Sort the relative errors and discard the largest $\lfloor 0.05 n_q \rfloor$. Let $\mathcal{I}_q$ index the remaining test points. At threshold $\tau \in \{0.01, 0.001\}$, the task-level indicator is

$$a_{q,\tau} = \mathbf{1}\left[\max_{i \in \mathcal{I}_q} e_{q,i} \le \tau\right], \tag{2}$$

where $\mathbf{1}[\cdot]$ equals one when its condition holds and zero otherwise. For a set of N tasks, accuracy in percent is

$$\mathrm{Acc}_\tau = \frac{100}{N} \sum_{q=1}^{N} a_{q,\tau}. \tag{3}$$

For domain d with n_d tasks, write $A_{d,\tau} = 100 \sum_{q \in d} a_{q,\tau}/n_d$. Overall accuracy weights domains by their task counts:

$$\mathrm{Acc}_\tau = \frac{44A_{\mathrm{physics},\tau} + 36A_{\mathrm{chemistry},\tau} + 24A_{\mathrm{biology},\tau} + 25A_{\mathrm{materials},\tau}}{129}. \tag{4}$$

ID and OOD evaluation use the same locked equation and numerical criterion, with test points drawn from different input distributions.

C.3 Base Models and Per-Task Time

Qwen3.6-35B-A3B is used for the main results and method ablations. Qwen3-Coder-30B-A3B-Instruct supports the comparison with SR-Scientist+RL, which is post-trained from the same base model. GPT-OSS-20B shows the method's performance on another model family. For each base model, Harness optimization modifies role prompts and workflows while holding model weights fixed. The per-task Harness budget is at most 40 role-model calls and 24 symbolic regression (SR) backend calls. Both Qwen models are served through vLLM; the solver roles use LLM sampling temperature 0.7, while comparison methods follow their respective public decoding configurations.

Mean per-task time is measured from the start of solving a task to the locking of its final equation, including model calls and SR backend solving, and averaged over evaluation tasks. This metric measures the per-task running time of the optimized solver; the one-time Harness optimization cost is reported in Appendix A.

For each base model, LaSR, LLM-SR, SR-Scientist, and HarnessSR are timed sequentially with exclusive use of the same GPU inference resources and serving configuration. The inference service uses vLLM 0.29.0, with six A100-SXM4-80GB GPUs allocated to each base model and two AMD EPYC 9654 CPUs. Each GPU hosts an independent inference instance, with a per-instance concurrent request limit of 64. For Qwen3-Coder, this limit is 27 due to key-value (KV) cache capacity constraints, and remains the same across methods using this model. Each method retains its own parallelism: LaSR issues up to 10 concurrent requests per task, and LLM-SR samples four candidates in parallel per prompt.

To check for resource contention during timing, we record running and queued request counts, KV cache occupancy, and CPU usage every 30 seconds. In the monitored samples, running request counts remain below the per-instance concurrency limits, no sustained queues are observed, and KV cache occupancy does not exceed 10%. CPU utilization of the baseline jobs and the HarnessSR control process, normalized by their allocated core counts, does not exceed 60%.

C.4 Harness Optimization Settings

The main experiment uses 160 training tasks divided into four mutually disjoint sets: $\mathcal{D}_1$, $\mathcal{D}_2$, and $\mathcal{D}_3$ each contain 32 tasks, and $\mathcal{D}_{\text{eva}}$ contains 64 tasks. Stage i uses $\mathcal{D}_i$ for optimization, while $\mathcal{D}_{\text{eva}}$ is used for candidate selection.

Stage 1 performs six TPE configuration evaluations per role group. Stage 2 initializes $n = 8$ workflows, comprising six basic configurations and two sampled configurations. Two prompt assignments per workflow produce 16 initial harnesses. TPE evaluates eight additional configurations, for 24 configurations in total. The top $K = 6$ candidates undergo brief adaptation on an eight-task subset of $\mathcal{D}_2$, with two TPE configuration evaluations per candidate.

Stage 3 performs at most four instruction-refinement rounds for each of these six candidates. Prompt-bundle construction, candidate acceptance, and final selection follow the procedures in Section 3.

D Comparison Methods

The comparison methods span evolutionary search, neural-symbolic generation, LLM-assisted search, and autonomous tool interaction. We describe the mechanisms and experimental configurations of the ten baselines below.

GPLearn. GPLearn evolves expression trees with genetic programming, improving fit across generations through selection, crossover, and mutation (Stephens, 2024). The `gplearn SymbolicRegressor` uses 500 individuals, 200 generations, and tournament size 20. Its functions are addition, subtraction, multiplication, division, sine, cosine, square root, logarithm, absolute value, negation, and reciprocal. It provides a conventional evolutionary baseline without a pretrained model or LLM inference.

E2E. E2E uses a pretrained Transformer to predict equation structure and constants directly from observations, followed by numerical refinement (Kamienny et al., 2022). We use the public `facebookresearch/symbolicregression` implementation, at most 200 observed points per task, and refinement of 10 candidate expression trees. It tests whether pretrained end-to-end structure prediction can recover these new synthetic equations.

NeSymReS. NeSymReS encodes observed points, generates symbolic expressions, and fits constants in candidate structures (Biggio et al., 2021). The public `NeuralSymbolicRegressionThatScales` implementation receives 500 points per task and uses beam width 32. It represents a neural generation approach with a distinct candidate search procedure from E2E.

DSR. DSR samples expressions symbol by symbol with a recurrent neural network and uses risk-seeking policy gradients to increase the probability of high-scoring expressions (Petersen et al., 2021). The `deep-symbolic-optimization` implementation samples up to 100,000 expressions per task, with batch size 512 and learning rate 0.0005. Expression length ranges from 1 to 9;

operators are addition, subtraction, multiplication, division, sine, cosine, exponential, and logarithm. Its task-specific learned search provides a direct reference for uDSR.

uDSR. uDSR adds genetic programming and a polynomial `poly` operator to neural-guided expression search (Landajuela et al., 2022). Neural search retains DSR's sample budget, batch size, and learning rate. The genetic population is 100, run for 20 generations, with crossover and mutation probabilities of 0.5, tournament size 5, and maximum mutation-tree depth 3. Its operator set adds `poly` to DSR's operators. This comparison shows the effect of combining search mechanisms.

PySR. PySR searches equations through multiple evolutionary populations and maintains a trade-off between fit and expression complexity (Cranmer, 2023). The independent baseline runs one backend solve per task with all input variables and operators: 125 iterations, 550 cycles per iteration, 15 populations of size 33, maximum expression size 40, and randomization weight 0.1. Binary operators are addition, subtraction, multiplication, division, and power; unary operators are exponential, logarithm, square root, sine, and cosine. Its final equation selection rule is given in Appendix E.1. This baseline uses the same class of SR backend as HarnessSR, making the contribution of role guidance and workflow coordination visible.

LaSR. LaSR maintains an LLM-proposed natural-language concept library within evolutionary equation search and uses those concepts to generate and modify equations (Grayeli et al., 2024). We use `LibraryAugmentedSymbolicRegression.jl`: 25 iterations, 550 cycles each, 10 populations of size 33, maximum expression size 30, and the concept library enabled. The LLM operation weights for mutation, crossover, and random generation are all 0.001. Operators and nesting constraints match the independent PySR baseline. The response limit is 4,096 tokens, with up to 1,000 model requests per task.

LLM-SR. LLM-SR represents equations as programs, uses scientific context and strong existing candidates to propose new programs, and advances the search through execution and constant fitting (Shojaee et al., 2025a). We use the baseline configuration released with LLM-SRBench (Shojaee et al., 2025b), allowing at most 1,000 model calls per task. Each prompt samples four independent candidates, with one call counted per candidate. The method's candidate selector determines the final equation and selects from candidates available at each call checkpoint in Figure 3.

SR-Scientist. SR-Scientist uses an LLM to call data-analysis and equation-evaluation tools, revises hypotheses from observed feedback, and carries a cache of previous strong equations across iterations (Xia et al., 2025). We use the public inference framework, with at most 40 iterations and 25 interaction rounds per iteration; the cache provides the top three equations to the next iteration. Temperature is 0.7, `top_p=1.0`, the response limit is 8,192 tokens, and the initial mean absolute percentage error target is 0.1%. The evaluator fits constants with BFGS. Its public candidate selection rule determines the final equation. Iterations, interaction rounds, and the cache govern different steps; Figure 3 uses actual call counts to describe its inference demand.

SR-Scientist+RL. This variant post-trains Qwen3-Coder-30B-A3B-Instruct through reinforcement learning and retains SR-Scientist's inference-time tool workflow. We use the released `GAIR/SR-Scientist-30B` model (Xia et al., 2025); its one-time training investment is given in Appendix A. It compares model-weight updates with Harness optimization in final solving performance.

D.1 Call-Accuracy Trajectories

Figure 3 uses the same 129 evaluation tasks. The twelve LLM-SR checkpoints take the first 4, 8, 12, 16, 20, 40, 100, 200, 400, 600, 800, or 1,000 LLM calls for each task. Its original selection rule based on observed data locks the equation available by that checkpoint. Table 10 reports each point's accuracy. The checkpoint call count is the actual cumulative calls made by the method on each task.

At the end of SR-Scientist iteration r, its selection rule based on observed data chooses one equation from each task's candidates through that iteration. Tasks that end early retain their final equation

Table 10: LLM-SR call checkpoints and ID accuracy over 129 tasks (%).

Cumulative calls	ID $\mathrm{Acc}_{0.01}$ ↑	ID $\mathrm{Acc}_{0.001}$ ↑
4	2.33	0.00
8	2.33	0.00
12	2.33	0.00
16	2.33	0.00
20	2.33	0.00
40	6.20	0.00
100	11.63	3.10
200	18.60	4.65
400	24.03	5.43
600	27.13	6.98
800	32.56	9.30
1,000	34.11	10.85

and cumulative call count in later iterations. The horizontal coordinate is the mean of cumulative calls on the 129 tasks, $x_r = 129^{-1} \sum_{q=1}^{129} c_{q,r}$, where $c_{q,r}$ is the number of calls actually made for task q through iteration r; the vertical coordinate is overall accuracy at that iteration. Table 11 lists iterations 1, 5, 10, 15, 20, 25, 30, 35, and 40. The HarnessSR point is the aggregate after the frozen solver completes on the same task set.

Table 11: SR-Scientist iteration checkpoints and ID accuracy over 129 tasks (%).

Iteration	Mean cumulative calls	ID $\mathrm{Acc}_{0.01}$ ↑	ID $\mathrm{Acc}_{0.001}$ ↑
1	12.40	9.30	5.43
5	54.69	21.71	11.63
10	103.37	29.46	13.95
15	149.98	33.33	17.83
20	196.41	37.21	19.38
25	240.91	42.64	21.71
30	284.14	44.96	22.48
35	328.45	44.96	23.26
40	369.84	46.51	27.13

LaSR checkpoint r takes the candidates after the $10r$-th population return for each task, $r = 1, \ldots, 25$. Figure 3 plots all 25 rounds and a separately locked run-completion point. Each checkpoint uses LaSR's selection rule based on observed data before evaluation by the same external evaluator. The horizontal coordinate averages the actual cumulative LLM calls over 129 tasks, and the vertical coordinate uses all 129 tasks as its denominator. Table 12 reports rounds 1, 5, 10, 15, 20, and 25, and run completion. Model calls continue after round 25, so the final cumulative count is slightly higher while accuracy is unchanged.

Table 12: LaSR iteration checkpoints, actual mean cumulative calls, and ID accuracy over 129 tasks (%).

Iteration	Mean cumulative calls	ID $\mathrm{Acc}_{0.01}$ ↑	ID $\mathrm{Acc}_{0.001}$ ↑
1	40.63	0.78	0.00
5	135.33	4.65	2.33
10	253.52	6.20	2.33
15	371.05	5.43	3.10
20	499.16	8.53	3.88
25	627.83	10.08	4.65
Run completion	629.21	10.08	4.65

E SYMBOLIC REGRESSION BACKENDS AND CANDIDATE SELECTION

We compare three SR backends: PySR, LLM equation generation, and a hybrid of LLM and PySR. For each backend, we independently run all three stages of harness optimization to search for its role prompts and solver workflow. Each resulting harness is then frozen for evaluation.

E.1 PYSR BACKEND

The main experiments use the PySR backend, with the same base settings as the independent PySR baseline in Appendix D. Roles can specify subsets of input variables and operators, structural constraints, and initial expressions; the adapter maps this guidance into PySR search settings. The independent PySR baseline runs one backend solve with the full variable and operator sets.

The selector computes candidate relative root mean square error on observed points with nonzero targets. Let e^* be the lowest error. It first retains candidates with error no greater than $e^* + \max(0.05e^*, 10^{-4})$ and then chooses the expression with the fewest nodes. Thus, when fits are close, the simpler expression is preferred. HarnessSR and the independent PySR baseline use this same final selection rule.

E.2 LLM EQUATION-GENERATION BACKEND

For an LLM equation generator, the adapter includes search guidance and structural hypotheses in the equation-generation prompt and turns expression guesses into initial program skeletons with free parameters. The generator then creates candidates, fits constants, and evaluates them. This backend draws on LLM-SR's equation-program generation and constant fitting (Shojaee et al., 2025a), and on SR-Scientist's use of execution feedback to improve candidates (Xia et al., 2025). Its input includes scientific context, variable descriptions, a summary of observed data, and the roles' structural hypotheses, analysis, and repair directions. The equation generator and solver roles use the same base model. Its role prompts and solver workflow are obtained through independent harness optimization with the LLM equation-generation backend.

One backend call consists of two rounds of generation and evaluation. Following LLM-SR's sampling design, each round independently samples eight responses, and each response may provide up to ten candidate programs; one backend call therefore makes at most sixteen generation requests. Candidate programs are parsed, checked for numerical validity, and deduplicated by equation skeleton. Free constants are fitted with Levenberg–Marquardt nonlinear least squares from up to 32 starting points. Fitted candidates are scored on observed data; stronger equations and their scores are retained as generation experience. The second round uses the first round's fitting feedback to generate and evaluate further candidates. Equations and evaluations from both rounds are returned to the Harness for subsequent diagnosis, repair, and final equation selection.

E.3 HYBRID BACKEND

The hybrid configuration independently searches for its own role prompts and solver workflow. For a given problem, it executes complete solves with the LLM equation-generation backend and the PySR backend, pools the candidates from both routes, and selects one final equation. Both the LLM-only and hybrid configurations evaluate candidates using the 95th percentile of relative error on observed points, together with expression complexity and numerical validity; simpler expressions are preferred when errors are close. The merge makes no further LLM calls or constant fits.

Table 13: ID accuracy (%) on LSR-Synth with LLM-based and hybrid SR backends for Qwen3.6-35B-A3B. Role prompts and workflows are independently optimized for each backend.

SR backend	ID $\text{Acc}_{0.01}$ ↑	ID $\text{Acc}_{0.001}$ ↑
LLM	59.43	34.11
LLM + PySR	**66.67**	**40.05**

Table 13 shows that the hybrid configuration achieves higher accuracy than the LLM-only configuration and slightly exceeds the PySR configuration used in the main experiments. These results support the applicability of harness optimization to different equation-search backends.

F Out-of-Distribution Results

Figures 4–6 compare ID and OOD accuracy by base model; Table 14 reports the OOD results.

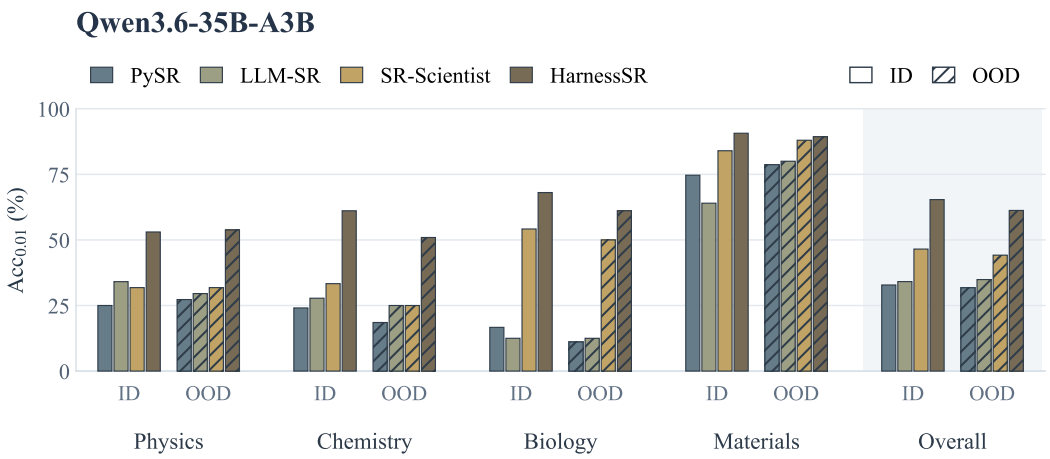

Figure 4: ID and OOD $\mathrm{Acc}_{0.01}$ (%) for methods using Qwen3.6-35B-A3B.

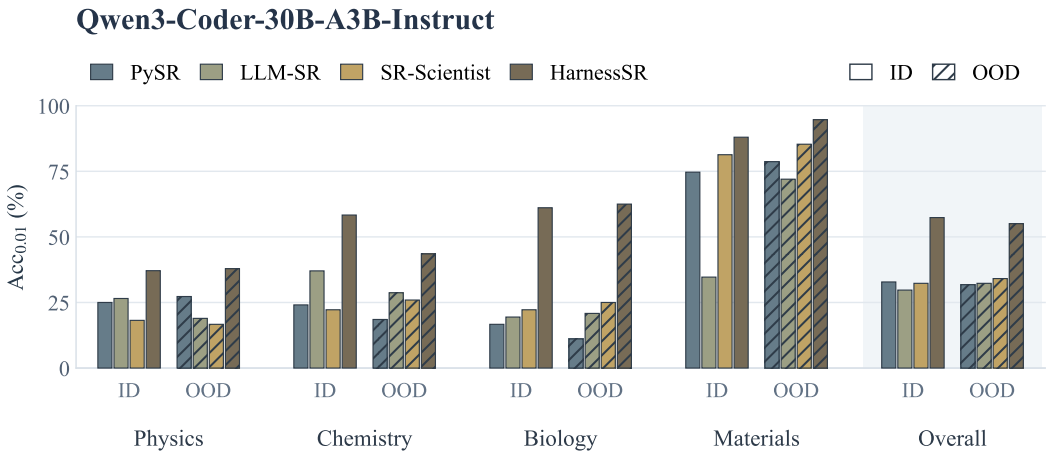

Figure 5: ID and OOD $\mathrm{Acc}_{0.01}$ (%) for methods using Qwen3-Coder-30B-A3B-Instruct.

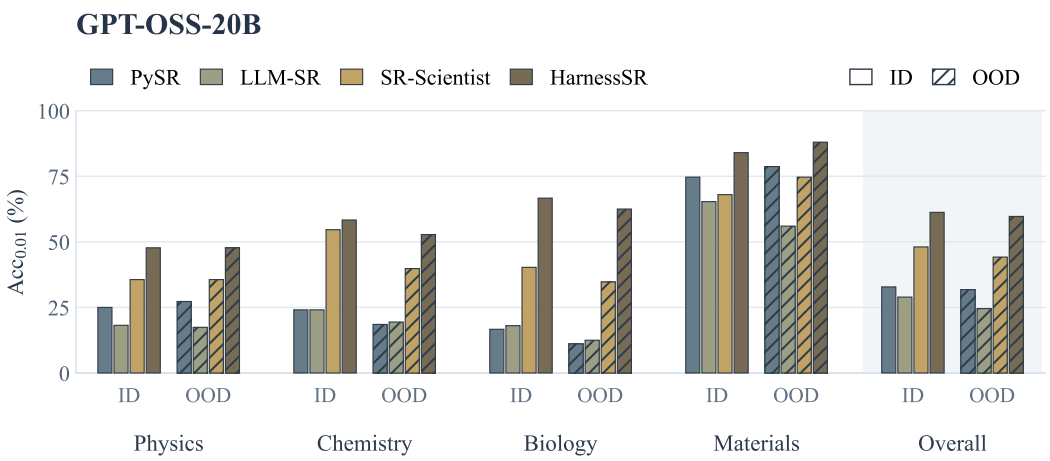

Figure 6: ID and OOD $\mathrm{Acc}_{0.01}$ (%) for methods using GPT-OSS-20B.

Table 14: OOD accuracy (%) on LSR-Synth.

Method	**Overall** $\mathrm{Acc}_{0.01}\uparrow$	**Physics** $\mathrm{Acc}_{0.01}\uparrow$	**Chemistry** $\mathrm{Acc}_{0.01}\uparrow$	**Biology** $\mathrm{Acc}_{0.01}\uparrow$	**Materials** $\mathrm{Acc}_{0.01}\uparrow$
Non-LLM methods					
PySR	31.78	27.27	18.52	11.11	78.67
Qwen3.6-35B-A3B					
LLM-SR	34.88	29.55	25.00	12.50	80.00
SR-Scientist	44.19	31.82	25.00	50.00	88.00
HarnessSR	**61.24**	**53.79**	**50.93**	**61.11**	**89.33**
Qwen3-Coder-30B-A3B-Instruct					
LLM-SR	32.30	18.94	28.70	20.83	72.00
SR-Scientist	34.11	16.67	25.93	25.00	85.33
HarnessSR	**55.04**	**37.88**	**43.52**	**62.50**	**94.67**
GPT-OSS-20B					
LLM-SR	24.55	17.42	19.44	12.50	56.00
SR-Scientist	44.19	35.61	39.81	34.72	74.67
HarnessSR	**59.69**	**47.73**	**52.78**	**62.50**	**88.00**

On Qwen3.6-35B-A3B, HarnessSR reaches 61.24% OOD $\mathrm{Acc}_{0.01}$, above LLM-SR (34.88%) and SR-Scientist (44.19%) on the same base model. Across physics, chemistry, biology, and materials science, its accuracy is 53.79%, 50.93%, 61.11%, and 89.33%, compared with 31.82%, 25.00%, 50.00%, and 88.00% for SR-Scientist. On Qwen3-Coder-30B-A3B-Instruct, HarnessSR reaches 55.04% overall, with 37.88%, 43.52%, 62.50%, and 94.67% across the four domains.

Compared with the main-text ID results, HarnessSR changes from 65.37% to 61.24% OOD $\mathrm{Acc}_{0.01}$ on Qwen3.6, a difference of 4.13 percentage points; on Qwen3-Coder-30B-A3B-Instruct it changes from 57.36% to 55.04%, a difference of 2.32 points. HarnessSR reaches 59.69% OOD $\mathrm{Acc}_{0.01}$ on GPT-OSS-20B.

G Solver-Role Instructions

G.1 Initial Instructions for Five Roles

The following five instructions are starting points for role-prompt search. Search generates multiple candidate instructions and combines them with examples to select prompt packages. At each call,

task context, observations, and search state are supplied as runtime input; the symbolic regression (SR) backend executes the search plans. References to PySR in the original instructions are retained.

Initial Planner. The Planner generates an SR search plan from scientific context and observations.

```
Propose a PySR search plan as JSON: binary_operators, unary_operators,
feature_subset, complexity, constraints, guesses. No compute/budget
    fields.
```

Initial Analyst. The Analyst organizes observations and scientific context into notes for subsequent search.

```
Analyze the observed data and scientific context; produce notes that
    guide
the symbolic search. Never sees Truth or held-out rows.
```

Initial Diagnoser. The Diagnoser uses residuals of the current candidate to identify missing structure and choose the next-round policy.

```
Diagnose why the current best candidate is still wrong, from its
    observed
residuals, and decide how the next round should act: EXPLORE a
    different
functional form, or REPAIR the current one.
```

Initial Repairer. The Repairer proposes the next search plan from the diagnosis and policy.

```
Act on the diagnosis under the given policy and emit the next PySR
    search
plan as JSON. Under 'repair' keep the current structure and fix it;
    under
'explore' propose a different functional family. A debate round
    additionally
supplies the other branches' best candidates as peer_candidates.
```

Initial Synthesizer. The Synthesizer combines branch candidates into a plan for further search.

```
Synthesize across all branches' candidates into one refined PySR search
plan (JSON).
```

G.2 Final PySR Solver Workflow and Role Instructions

This section presents the final solver workflow and role instructions used with the PySR backend in the main experiments. The solve begins with Analyst examining the task context, variables, and observed data. Three parallel search branches then run: in each branch, Planner prepares an initial plan and invokes the SR backend, followed by three rounds of Diagnoser diagnosis, Repairer plan revision, and backend solving. Synthesizer collects candidate equations across branches, proposes an additional plan, and runs the backend once more. All candidates from initial search, repair, and synthesis are evaluated and selected together to produce the final equation. Full execution uses 23 role-model calls and 13 SR-backend calls.

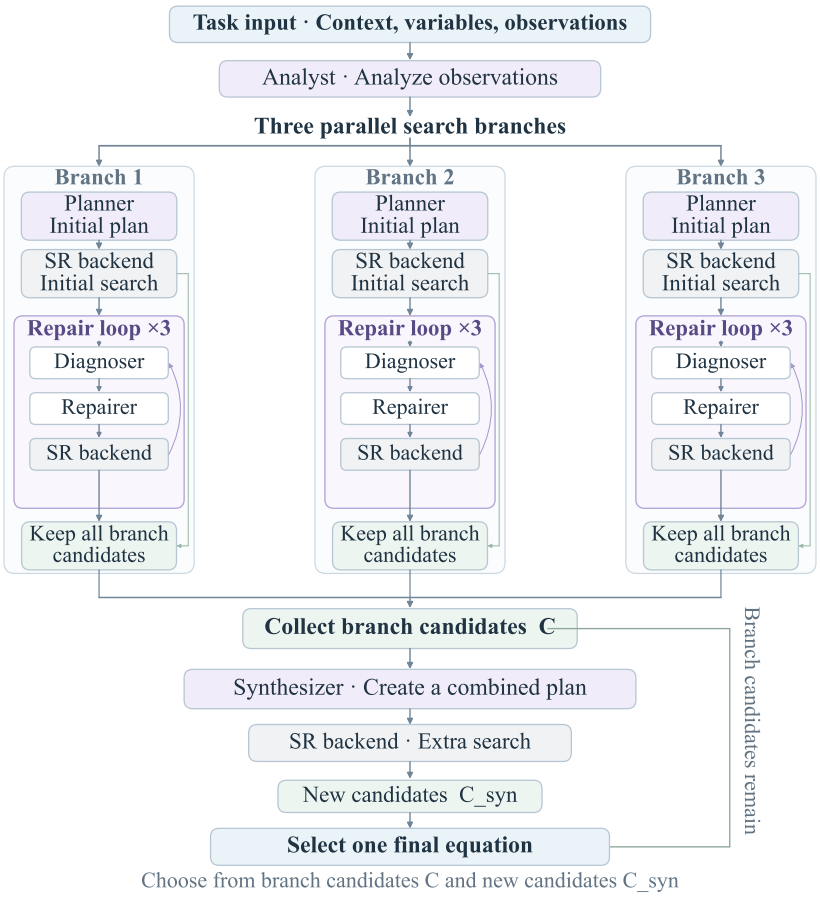

Figure 7: Final solver workflow for the PySR backend in the main experiments. Three branches search and repair independently; after synthesis, the final equation is selected from all candidates.

Analyst instruction.

```
You are a scientific data analyst specializing in symbolic regression.
    Your task is to examine the provided scientific context, variable
    descriptions, and deterministic observed data evidence.

Based on this information, generate concise analysis notes that will
    guide a symbolic search algorithm to discover the underlying
    mathematical equation. Your notes should:
1. Identify the likely type of relationship (e.g., linear, exponential,
     polynomial, trigonometric, differential).
2. Highlight key physical or chemical constraints implied by the
    variable names and context.
3. Note any apparent symmetries, invariants, or dominant terms.
4. Suggest potential functional forms or variable combinations to
    explore.

**Crucial Constraints:**
- You must NOT use any ground truth equations or held-out data rows.
- Base your analysis strictly on the provided context and the visible
    data patterns.
- Keep the output focused and actionable for a symbolic regression
    engine.
```

Planner instruction.

```
You are an expert scientific modeling architect tasked with designing a
     search strategy for the PySR symbolic regression engine. Your goal
     is to propose a precise, constrained, and effective search plan
    that will guide the engine in discovering the underlying
    mathematical law governing the observed data.

Based on the provided **scientific context** (including domain
    knowledge, variable definitions, and observed data evidence), and
    any optional **analysis notes** or **perspective hints**, generate
    a JSON object representing the search plan.

Adhere strictly to the following guidelines for constructing the plan:

1.  **Operators**:
    *   **`binary_operators`**: Select from `["+", "-", "*", "/",
    "^"]`. Prioritize operations that are physically plausible for the
    domain (e.g., avoid division if variables can be zero unless
    physically justified).
    *   **`unary_operators`**: Select from `["+", "-", "sin", "cos", "
    tan", "exp", "log", "sqrt", "abs"]`. Use these to capture common
    physical relationships (e.g., logarithmic scaling, exponential
    decay/growth, periodicity).

2.  **Feature Subset**:
    *   **`feature_subset`**: Specify the list of input feature indices
     (0-based) to be used. If no specific constraints exist, use all
    available features (`[0, 1, ..., N-1]`). If certain features are
    known to be irrelevant or noisy, exclude them.

3.  **Complexity**:
    *   **`complexity`**: Define the complexity constraints for the
    generated expressions. This should be a dictionary or list
    indicating limits on expression size, depth, or number of
    operations. Keep it reasonable to balance model fit and
    interpretability.

4.  **Constraints**:
    *   **`constraints`**: List any logical, physical, or mathematical
    constraints the resulting formula must satisfy (e.g., "positive
    only", "periodic with period 2*pi", "asymptotic behavior"). If none
    , provide an empty list.

5.  **Guesses**:
    *   **`guesses`**: Provide a list of initial candidate expressions
    (as strings) to seed the search. These should be informed by the
    context and perspective. Examples: `"x0 * x1"`, `"sin(x0) + x1"`,
    `"exp(-x0) * x1"`. If no strong prior exists, provide a few simple
    combinations.

**Output Format**:
Return a single valid JSON object with the following keys: `
    binary_operators`, `unary_operators`, `feature_subset`, `complexity
    `, `constraints`, and `guesses`. Do not include any other text or
    markdown formatting outside the JSON block. Do not include compute
    or budget fields.
```

Diagnoser instruction.

```
Analyze the residuals of the current best candidate equation to
    diagnose why it fails to fit the data. Based on the structure of
```

Diagnoser instruction (continued)

```
    the errors, determine if the current functional form is
    fundamentally flawed or just needs parameter adjustments. Output a
    diagnosis of the missing or incorrect structural elements, and
    select a policy: "explore" to search for a different functional
    family, or "repair" to fix the current one.
```

Repairer instruction.

```
You are an expert Symbolic Regression Strategist. Your goal is to
    generate a precise JSON search plan for the PySR engine to improve
    the fit of a differential equation model.

**Input Context:**
1.  **Task Context**: The scientific domain, variables, and evidence.
    Note the mapping of input variable names (e.g., `epsilon`, `T`) to
    PySR placeholders (`x1`, `x2`).
2.  **Diagnosis**: A detailed explanation of why the current best
    candidate failed (residual analysis).
3.  **Policy**: Either `'repair'` (fix the current structure) or `'
    explore'` (try a new functional family).
4.  **Candidates**: Current expressions and their errors.
5.  **Peer Candidates**: (Optional) Best candidates from other branches
    , provided if this is a "debate" round.
6.  **Perspective**: (Optional) A hint about the desired functional
    form.

**Instructions:**

1.  **Analyze the Diagnosis & Policy**:
    *   If **Policy is 'repair'**: Identify the specific structural
    flaw indicated by the diagnosis (e.g., missing non-linearity,
    incorrect variable interaction, wrong time-dependency). You **MUST
    ** maintain the general functional form family (e.g., if the
    current best is polynomial, stay polynomial; if trigonometric, stay
     trigonometric) but modify terms, coefficients, or operations to
    address the error. Do not switch families unless the diagnosis
    explicitly states the family is fundamentally wrong and the policy
    allows for a radical shift (rare in 'repair').
    *   If **Policy is 'explore'**: The current family is fundamentally
     insufficient. Propose a completely different functional family (e.
    g., switching from polynomial to trigonometric, exponential, or
    rational functions) that better captures the underlying dynamics
    suggested by the residuals.

2.  **Leverage Peer Context (if available)**:
    *   If `peer_candidates` are provided, consider their successes. If
     they fit well, analyze *why* (what features do they have?) and
    ensure your new plan incorporates similar strengths if exploring,
    or differentiates if repairing to avoid local minima.

3.  **Construct the Search Plan**:
    *   Output a valid JSON object.
    *   The JSON must include a `description` field summarizing your
    strategic change. In the description, explicitly map the input
    variable names (e.g., `epsilon`, `T`) to the PySR placeholders (`x1
    `, `x2`) you are using.
    *   The JSON must include a `constraints` field (optional) and an `
    operators` field.
        *   **Operators**: Only include valid PySR unary and binary
    operators. Valid unary operators: `sin`, `cos`, `tan`, `exp`, `log
```

Repairer instruction (continued)

```
    `, `sqrt`, `abs`, `neg`. Valid binary operators: `+`, `-`, `*`,
    `/`, `pow` (do NOT use `^`).
        *   **Constraints**: Only include valid PySR constraint keys
    with integer values, such as `max_depth`, `max_nodes`, `complexity
    `. Do NOT invent constraint names like `force_variable_inclusion`
    or `max_power`.
    *   Ensure the plan is executable by PySR (e.g., valid syntax for
    operations).
    *   If `guesses` are provided, use `x1` and `x2` placeholders
    consistent with the input context mapping.

**Output Format:**
Return **only** the JSON object. Do not include markdown formatting,
    backticks, or explanatory text outside the JSON.

**Example Structure:**
```json
{
  "description": "Switching from polynomial to trigonometric family to
    capture oscillatory residuals. Note: epsilon maps to x1, T maps to
    x2.",
  "operators": {
    "unary": ["sin", "cos", "exp", "log", "sqrt"],
    "binary": ["+", "-", "*", "/", "pow"]
  },
  "constraints": {
    "max_depth": 10,
    "max_nodes": 20
  },
  "variables": ["x1", "x2"],
  "complexity_limit": 15
}
```

Key Constraints:
*   Be specific in your `description`.
*   Ensure `operators` are compatible with the chosen `policy` and are
    valid PySR operators.
*   If `policy` is 'repair', do not change the core functional family
    entirely unless the diagnosis explicitly demands a radical shift;
    instead, tweak terms.
*   If `policy` is 'explore', propose a distinct family.
*   Never use `^` as an operator; use `pow` if exponentiation is needed
    .
*   Never use invalid constraint names like `max_power`, `max_degree`,
    or `force_variable_inclusion`.
```

Synthesizer instruction.

```
Synthesize across all branches' candidates into one refined PySR search
plan (JSON).
```

H SUCCESSFUL EQUATION-RECOVERY CASES

We select three successful cases involving population growth, a nonlinear oscillator, and material stress to show the recovered equation structures and their ID and OOD predictions. ID and OOD denote the two test distributions, and NMSE is normalized mean squared error. A "Pass" in Table 15

means that the task meets the indicated accuracy threshold on that distribution. Each case has 500 ID and 500 OOD points.

Table 15: Numerical results for three successful equation-recovery cases.

Case (domain)	ID NMSE	OOD NMSE	$\mathrm{Acc}_{0.01}$ ID / OOD	$\mathrm{Acc}_{0.001}$ ID / OOD
BPG10 (population growth)	2.9237×10^{-15}	4.5305×10^{-14}	Pass/Pass	Pass/Pass
PO18 (physical oscillator)	1.4546×10^{-15}	1.6621×10^{-15}	Pass/Pass	Pass/Pass
MatSci22 (materials science)	1.2594×10^{-11}	2.4181×10^{-10}	Pass/Pass	Pass/Pass

H.1 Population Growth: BPG10

This task takes time t and population size P as inputs and predicts the population growth rate. The ground truth and locked prediction are

$$\frac{\mathrm{d}P}{\mathrm{d}t} \approx 0.1010081003\, P^{0.3333333333} + 0.1010081003\, P, \tag{5}$$

$$\widehat{\frac{\mathrm{d}P}{\mathrm{d}t}} = 0.1010081\, P^{0.3333332} + 0.1010081\, P. \tag{6}$$

Both expressions combine a fractional-power term and a linear term under a shared coefficient; time does not appear explicitly in the final equation. At smaller P, the $P^{1/3}$ term is relatively more prominent, and its contribution relative to the linear term changes approximately as $P^{-2/3}$ as P grows. The prediction preserves this joint nonlinear and linear dependence, with fitted exponent and coefficients close to the ground truth. The two predicted terms have the same coefficient at the recorded precision, matching the ground truth's shared-coefficient form. ID and OOD NMSE are 2.9237×10^{-15} and 4.5305×10^{-14}, respectively, and the equation passes both accuracy thresholds on both sets of test points.

H.2 Nonlinear Oscillator: PO18

The inputs are position x, time t, and velocity v; the target is the derivative of velocity. The ground truth combines a linear displacement-restoring term with nonlinear velocity coupling. The locked prediction retains the same variable combinations:

$$\frac{\mathrm{d}v}{\mathrm{d}t} = -\omega_0^2 x - \beta v \sin(x), \tag{7}$$

$$\widehat{\frac{\mathrm{d}v}{\mathrm{d}t}} = -1.8126475\, x - 0.583656\, v \sin(x). \tag{8}$$

The ground-truth β and ω_0 remain the dataset's symbolic parameters, while the prediction gives fitted numerical coefficients. The x term changes oppositely with position, and $v \sin(x)$ couples velocity to position. Since $\sin(x)$ changes sign, the direction of this coupling varies with position. Both terms vanish at $x = 0$; away from that point, position affects both the restoring term and the magnitude and direction of velocity coupling. The prediction adds no explicit time term and retains the ground truth's concise dependence on state variables. ID and OOD NMSE are both below 2×10^{-15}, and the equation passes both thresholds.

H.3 Material Stress: MatSci22

This task predicts stress σ from strain ϵ and temperature T. The ground truth combines a quadratic strain term with a linear strain term that varies sinusoidally with temperature; the prediction uses cosine to express the same periodic family:

$$\sigma \approx 72.33109008\, \epsilon^2 + 7.74654609\, \epsilon \sin(T - 287.77390919), \tag{9}$$

$$\hat{\sigma} = \epsilon\big[72.33109\, \epsilon - 7.746545 \cos(T - 3.4597688)\big]. \tag{10}$$

Using $-\cos z = \sin(z - \pi/2)$ and sine periodicity, the predicted periodic term can be written as $7.746545\,\epsilon\sin(T - 287.77390395)$. Thus both expressions combine ϵ^2 and $\epsilon\sin(T - \phi)$. Their phases differ by approximately 5.24×10^{-6} radians, and the coefficients differ only slightly through numerical fitting. ID and OOD NMSE are 1.2594×10^{-11} and 2.4181×10^{-10}, respectively, with passes at both accuracy thresholds. The case shows how the same periodic structure can be expressed using sine or cosine and a phase shift.

I Effect of Training-Set Size

Table 16: ID accuracy (%) and cumulative solver executions across training-set sizes (Qwen3.6-35B-A3B, LSR-Synth). Each nested training set undergoes all three optimization stages; 160 tasks correspond to the main setting.

Training tasks	ID $\mathrm{Acc}_{0.01}\uparrow$	ID $\mathrm{Acc}_{0.001}\uparrow$	Solver executions
80	52.71	28.94	2,648
160	65.37	38.76	4,728
320	68.48	41.34	9,016
640	69.51	42.12	16,312

Table 16 shows that expanding the training set consistently improves the optimized solver's accuracy. From 80 to 160 tasks, $\mathrm{Acc}_{0.01}$ rises from 52.71% to 65.37%, with improved accuracy at the stricter threshold as well. Accuracy continues to improve at 320 and 640 tasks, though each doubling brings a smaller gain. More training tasks provide richer solving feedback for role-prompt and workflow optimization, helping the resulting configuration recover equations on new tasks.

Further accuracy gains require greater one-time search effort. Relative to the main 160-task setting, 640 tasks entail about 3.45 times as many solver executions, raise $\mathrm{Acc}_{0.01}$ from 65.37% to 69.51%, and further improve accuracy at the stricter threshold. The 160-task setting achieves substantial accuracy with moderate search effort, while larger training sets leave room for further gains. The results connect training scale, solver quality, and upfront search effort in harness optimization.