EGG-SR: EMBEDDING SYMBOLIC EQUIVALENCE INTO SYMBOLIC REGRESSION VIA EQUALITY GRAPH

Anonymous authors
Paper under double-blind review

ABSTRACT

Symbolic regression seeks to uncover physical laws from experimental data by searching for closed-form expressions, which is an important task in AI-driven scientific discovery. Yet the exponential growth of the search space of expression renders the task computationally challenging. A promising yet underexplored direction for reducing the effective search space and accelerating training lies in *symbolic equivalence*: many expressions, although syntactically different, define the same function – for example, $\log(x_1^2x_2^3)$, $\log(x_1^2) + \log(x_2^3)$, and $2\log(x_1) + 3\log(x_2)$. Existing algorithms treat such variants as distinct outputs, leading to redundant exploration and slow learning. We introduce EGG-SR, a unified framework that integrates equality graphs (e-graphs) into diverse symbolic regression algorithms, including Monte Carlo Tree Search (MCTS), deep reinforcement learning (DRL), and large language models (LLMs). EGG-SR compactly represents equivalent expressions through the proposed EGG module, enabling more efficient learning by: (1) pruning redundant subtree exploration in EGG-MCTS, (2) aggregating rewards across equivalence classes in EGG-DRL, and (3) enriching feedback prompts in EGG-LLM. Under mild assumptions, we show that embedding e-graphs tightens the regret bound of MCTS and reduces the variance of the DRL gradient estimator. Empirically, EGG-SR consistently enhances a class of modern symbolic regression algorithms across multiple benchmarks, discovering equations with lower normalized mean squared error.

1 INTRODUCTION

Symbolic regression aims to automatically discover physical laws from experimental data and has been widely used in scientific domains (Schmidt & Lipson, 2009; Udrescu & Tegmark, 2020; Cory-Wright et al., 2024; LaFollette et al., 2025; La Cava et al., 2021). Many contemporary methods for symbolic regression formulate the search for optimal expressions as a sequential decision-making process. In literature, existing models learn to predict the optimal sequence of grammar rules (Sun et al., 2023), the traversal sequence for expression trees (Petersen et al., 2021; Kamienny et al., 2022), or executable strings that follow Python syntax (Shojaee et al., 2025; Zhang et al., 2025). This task remains computationally challenging due to its NP-hard nature (Virgolin & Pissis, 2022), that is, the search space of candidate expressions grows exponentially with the data dimension.

A promising yet underexplored direction for reducing the search space and accelerating discovery is the integration of *symbolic equivalence* into learning algorithms. For example, these expressions $\log(x_1^2x_2^3)$, $\log(x_1^2)+\log(x_2^3)$, and $2\log(x_1)+3\log(x_2)$ all represent the same math function and are therefore *symbolically equivalent*. Ideally, a well-trained model would recognize such equivalences and assign identical probabilities, rewards, or losses to the corresponding predicted expressions (Allamanis et al., 2017), since these expressions produce identical functional outputs and attain the same prediction error on the dataset. In the literature, existing algorithms treat these expressions as distinct outputs, leading to redundant exploration of the search space and slow convergence. The main challenge of this direction is: how to represent equivalent expressions and embed them into modern learning models in a unified and scalable manner?

Since the number of equivalent variants grows exponentially with expression length, explicitly representing the full set of equivalent expressions quickly becomes both time-consuming and memory-intensive. To address this challenge, a line of recent works introduced the equality graph (e-graph), a

data structure that compactly encodes the set of equivalent variants by storing shared sub-expressions only once (Nandi et al., 2021; Willsey et al., 2021; Kurashige et al., 2024). E-graphs have since been successfully applied to diverse tasks, including program optimization (Barbulescu et al., 2024), dataset generation of equivalent expressions (Zheng et al., 2025). In genetic programming-based symbolic regression, E-graphs have been used for duplicate detection (de França & Kronberger, 2025), expression simplification (de França & Kronberger, 2023), TopK query and pattern matching (de França & Kronberger, 2025). Despite these empirical successes, we find that a unified framework that enables diverse symbolic regression algorithms to interact with e-graphs for accelerating the discovery has room for improvement.

We present a unified framework, EGG-SR, that integrates symbolic equivalence into learning algorithms using e-graphs, to accelerate a broad range of symbolic regression methods. Our framework encompasses EGG-enhanced Monte Carlo Tree Search (EGG-MCTS), EGG-enhanced Deep Reinforcement Learning (EGG-DRL), and EGG-enhanced Large Language model (EGG-LLM). The core idea is to leverage e-graphs to efficiently sample multiple equivalent variants of expressions and compute a new equivalence-aware learning objective. Specifically, (1) in search tree–based methods, EGG-MCTS prunes redundant exploration over equivalent subtrees; (2) in reward-driven learning, EGG-DRL aggregates rewards over equivalent expressions, stabilizing training; (3) in LLM-based approaches, EGG-LLM enriches the feedback prompt with multiple equivalent expressions to better guide next round generation. Under mild theoretical assumptions, we show the benefit of embedding symbolic equivalence into learning: (1) EGG-MCTS offers a tighter regret bound than standard MCTS (Sun et al., 2023), and (2) The gradient estimator of EGG-DRL exhibits a lower variance than that of standard DRL (Petersen et al., 2021).

In experiments, we evaluate EGG-SR with a wide class of symbolic regression baselines across several challenging benchmarks. We demonstrate its advantages over existing approaches using EGG than without. The e-graph module consistently improves performance across diverse frameworks, including MCTS, DRL, and LLM. EGG helps to discover symbolic expressions with lower normalized mean squared error than baseline methods.

2 PRELIMINARIES

Symbolic Expression. Let $\mathbf{x} = (x_1, \ldots, x_n)$ denote input variables and $\mathbf{c} = (c_1, \ldots, c_m)$ be coefficients. A symbolic expression ϕ connects these variables and coefficients using mathematical operators such as addition, multiplication, and logarithms. For example, $\phi = 3\log x_1 + 2\log x_2$ is a symbolic expression composed of variables $\{x_1, x_2\}$, operators $\{+, \log\}$, and coefficients $\{c_1 = 3, c_2 = 2\}$. Symbolic expressions can be represented as binary trees (de França & Kronberger, 2025), pre-order traversal sequences of the tree (Petersen et al., 2021), topological traversal sequences of expression graph (Kahlmeyer et al., 2024; Xiang et al., 2025), or sequences of production rules defined by a context-free grammar (Sun et al., 2023).

To uniformly handle all symbolic objects in this work, we use a context-free grammar to represent symbolic expressions. The grammar is defined by the tuple $\langle V, \Sigma, R, S\rangle$ where (1) V is a set of non-terminal symbols representing arbitrary sub-expressions, such as $V = \{A\}$; (2) Σ is a set of terminal symbols, including input variables and coefficients, i.e., $\{x_1, \ldots, x_n, \mathsf{c}\}$; (3) R is a set of production rules representing mathematical operations, such as addition or multiplication. For example, $A \to A \times A$ denotes multiplication, and the semantics is to replace the left-hand side with the right-hand side; (4) S is the start symbol, typically set to $S = A$. By applying a sequence of production rules from the start symbol, each rule sequentially replaces the first non-terminal symbol in the expression. The resulting expression, free of non-terminals, corresponds to a valid expression.

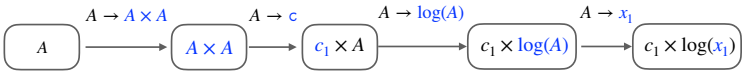


Figure 1: Transforming a sequence of grammar rules into a valid expression. At each step, the *first* non-terminal symbol inside the squared box is expanded. The expanded parts are color-highlighted.

Figure 1 visualize an example rule sequence $(A \to A \times A, A \to \mathsf{c}, A \to \log(x_1))$. The first multiplication rule $A \to A \times A$ expands the start symbol $\phi = A$ to $\phi = A \times A$. Applying the second rule $A \to \mathsf{c}$ yields $\phi = c_1 \times A$. We assign an index to the coefficient symbol c, to differentiate multiple coefficients. Finally, we apply $A \to \log(x_1)$ to $\phi = c_1 \times A$, and obtain $\phi = c_1 \log(x_1)$.

Symbolic Equivalence under a Rewrite System. Rewrite rules are employed to simplify, rearrange, and reformulate expressions in tasks, such as code optimization and automated theorem proving (Huet & Oppen, 1980; Nandi et al., 2021). A rewrite system provides a systematic procedure for transforming expressions by replacing sub-expressions according to predefined patterns.

Formally, a rewrite rule r_i is written as "LHS $\rightsquigarrow$ RHS", where the left-hand side (LHS) specifies a match pattern and the right-hand side (RHS) specifies its replacement. Given a set of rules $\mathcal{R} = \{r_1, \ldots, r_L\}$, the *symbolic equivalence* relation $\equiv_{\mathcal{R}}$ induced by $\mathcal{R}$ is defined as follows: for two symbolic expressions ϕ_1 and ϕ_2,

$$\phi_1 \equiv_{\mathcal{R}} \phi_2 \qquad \Longleftrightarrow \qquad \phi_1 \to^* \phi_2 \text{ or } \phi_2 \to^* \phi_1, \tag{1}$$

where $\phi_1 \to^* \phi_2$ denotes that ϕ_1 can be transformed into ϕ_2 through a finite sequence of applications of rules in $\mathcal{R}$. In other words, two expressions are equivalent under $\mathcal{R}$ if one can be rewritten into the other by repeated application of these rules. In this work, we systematically encode known mathematical identities as rewrite rules $\mathcal{R}$ to generate equivalent variants of symbolic expressions.

For example, consider the rewrite rule $r_1 = \log(a \times b) \rightsquigarrow \log(a) + \log(b)$, where a and b are placeholders for arbitrary sub-expressions. In our grammar-based representation, LHS corresponds to the rule sequence $(A \to \log(A),\ A \to A \times A,\ A \to a,\ A \to b)$, and RHS corresponds to $(A \to A + A,\ A \to \log(A),\ A \to a,\ A \to \log(A),\ A \to b)$. Figure 1 gives a visualized interpretation of this representation. Applying r_1 to $\phi_1 = \log(x_1^3 x_2^2)$ yields $\phi_2 = \log(x_1^3) + \log(x_2^2)$. Thus ϕ_1 and ϕ_2 are *symbolically equivalent* under r_1.

Symbolic Regression searches for a closed-form expression from the experimental data, which are widely applied in diverse scientific domains (Ma et al., 2022). Given a dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, the task searches for the optimal expression ϕ^*, that minimizes the averaged loss on the dataset:

$$\phi^* \leftarrow \arg\min_{\phi \in \Phi} \frac{1}{N} \sum\nolimits_{i=1}^{N} \ell(\phi(\mathbf{x}_i, \mathbf{c}), y_i),$$

where Φ denotes the search space containing all expressions; the loss function ℓ measures the distance between the output of the candidate expression $\phi(\mathbf{x}_i, \mathbf{c})$ and the ground truth y_i. The coefficient $\mathbf{c}$ within expression ϕ are optimized on training data D using optimizers, like BFGS (Fletcher, 2000). Since the search space of expressions Φ is exponentially large in the data dimension, finding the optimal expression is challenging. In fact, this task has been proven NP-hard (Virgolin & Pissis, 2022). Recent works to mitigate this challenge are reviewed in section 4.

3 Methodology

3.1 EGG: Equality graph for Grammar-based Symbolic Expression

Enumerating all equivalent variants of a symbolic expression is combinatorially expensive. Storing these variants explicitly (e.g., in an array) is highly space-inefficient. To mitigate this challenge, we adopt the recently proposed Equality graph (i.e, e-graph) data structure (Willsey et al., 2021; Waldmann et al., 2022), which compactly represents the set of equivalent expressions by sharing common subexpressions. We extend e-graphs to support grammar-based symbolic expressions, noted as EGG, facilitating seamless integration with symbolic regression algorithms.

Definition. An *e-graph* consists of a collection of equivalence classes, called *e-classes*. Each e-class contains a set of *e-nodes* representing symbolically equivalent sub-expressions (Willsey et al., 2021). Each e-node encodes a mathematical operation and references a list of child e-classes corresponding to its operands. Edges always point from an e-node to e-classes.

Figure 2(d) shows an example e-graph. The color-highlighted part is an e-class (in dashed box) containing two e-nodes (in solid boxes): "$A \to \log(A)$" and "$A \to A + A$". The two e-nodes represent logarithmic operation and addition, respectively. The e-node "$A \to \log(A)$" has a single edge to its child e-class "$A \to A \times A$", because log-operator involves one operand.

Construction. An e-graph is initialized with a sequence of production rules, representing an input expression. A predefined set of mathematical laws is formatted as rewrite rules (as defined in section 2). Each rule is applied by *matching* its left-hand side (LHS) pattern against the current e-graph, which involves traversing all e-classes to identify subexpressions that match LHS. For every

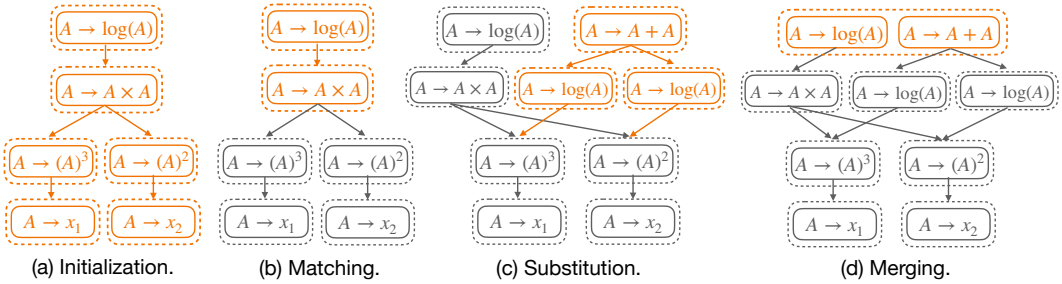


Figure 2: Applying the rewrite rule $\log(a \times b) \rightsquigarrow \log(a) + \log(b)$ to an e-graph representing expression $\log(x_1^3 x_2^2)$. **(a)** The initialized e-graph consists of *e-classes* (dashed boxes), each containing equivalent *e-nodes* (solid boxes). Edges connect e-nodes to their child e-classes. **(b)** The matching step identifies the e-nodes that match the `LHS` of the rule. **(c)** The substitution step adds new e-classes and edges corresponding to the `RHS` to the e-graph. **(d)** The merging step consolidates equivalent e-classes. The final e-graph in (d) compactly represents two equivalent expressions. Section 3.1 provide a detailed definition of each procedure.

successful match, new e-classes and e-nodes corresponding to the right-hand side (`RHS`) of the rule are created. A *merge* operation is applied to incorporate the new e-class into the matched e-class, thereby preserving the structure of known equivalences. This process, known as *equality saturation*, iteratively applies pattern matching and merging until either no further rules can be applied or a maximum number of iterations is reached.

Figure 2 shows an example of e-graph construction with the rewrite rule $\log(a \times b) \rightsquigarrow \log(a) + \log(b)$, where a and b are placeholders for arbitrary sub-expressions. The e-graph in initialized with an expression $\phi = \log(x_1^3 x_2^2)$ in Figure 2(a). The `LHS` is matched against the color-highlighted e-classes in Figure 2(b), where $a = x_1^3$ and $b = x_2^2$. The substitute step first constructs e-classes and e-nodes that represent `RHS`, which are color-highlighted in Figure 2(c). One e-class is merged with the matched e-class in Figure 2(d). The resulting e-graph represents the set of two equivalent expressions: $\log(x_1^3 x_2^2)$ and $\log(x_1^3) + \log(x_2^2)$. The e-graph in Figure 2(d) is space-efficient, because two sub-expressions x_1^3 and x_2^2 are stored only once in the e-graph. Additional EGG visualizations on more complex expressions—including those involving trigonometric and partial-derivative rewrite rules—are provided in Appendix F.1.

Extraction. After the e-graph is saturated, an extraction step is performed to obtain K representative expressions (Goharshady et al., 2024). Because an e-graph encodes up to an exponential number of equivalent expressions, exhaustive enumeration is computationally infeasible. We therefore adopt two practical strategies: (1) *cost-based extraction*, which selects several simplified expressions by minimizing a user-defined cost function over operators and variables, and (2) *random-walk sampling*, which generates a batch of expressions by stochastically traversing the e-graph. The detailed procedure is provided in Appendix A.4.

Implementation. Existing e-graph implementations are available in Rust (Willsey et al., 2021), Haskell (Hegg or srtree library), and Julia language (Metatheory.jl) or provided as Python wrappers (Shanabrook, 2024; de França & Kronberger, 2025), neither of which is well-suited for grammar-based symbolic expression. We therefore implement EGG in pure Python without any dependency on Rust or Haskell, following the algorithmic pipeline in Willsey et al. (2021). Our implementation supports grammar-based expressions and integrates seamlessly with the sequential decision-making process, which is essential for integration with modern learning algorithms. EGG incorporates a wide class of rewrite rules, including those for logarithmic, trigonometric, and differential operators. Implementation details are provided in Appendix A.

3.2 EGG-SR: EMBED SYMBOLIC EQUIVALENCE INTO SYMBOLIC REGRESSION VIA EGG

Embedding EGG into Monte Carlo Tree Search. MCTS (Sun et al., 2023) maintains a search tree to explore optimal expressions. Each edge is labeled with a production rule, and the label of a node is determined by the sequence of edge labels from the root. By construction, the node label corresponds to either a valid or a partially complete expression. A node representing a partially complete expression can be further expanded by extending its children using production rules.

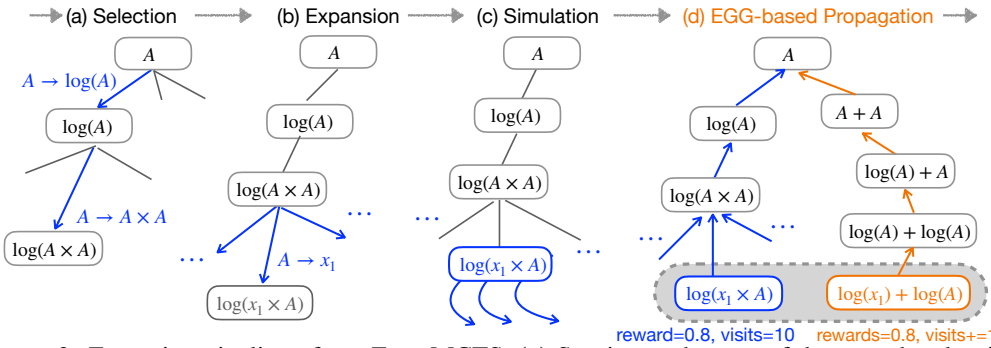


Figure 3: Execution pipeline of our EGG-MCTS. **(a)** Starting at the root of the tree, the algorithm selects the child with the highest UCT score (in equation 2) until reaching a leaf. **(b)** The selected leaf is expanded by applying all applicable production rules, producing new child nodes. **(c)** For each child, several simulations are run to complete the expression template by sampling additional rules. The resulting expressions are fitted to the data to estimate their coefficients. **(d)** Rewards and visit counts from evaluated children are back-propagated up the tree. Updates are applied to the selected and also equivalent paths (highlighted in two colors), enabled by our EGG module.

During learning, the MCTS algorithm executes the following four steps (Brence et al., 2021): (1) *Selection*. Starting from the root node, successively select the production rule with the highest Upper Confidence Bound for Trees (UCT) (Kocsis & Szepesvári, 2006) to reach a leaf node s:

$$\texttt{UCT}(s,a) = \texttt{Reward}(s,a) + \alpha\sqrt{\log(\texttt{visits}(s))/\texttt{visits}(s,a)} \tag{2}$$

where $\texttt{Reward}(s,a)$ is the average reward obtained by applying production rule a at node s, $\texttt{visits}(s)$ is the total visits to node s, and $\texttt{visits}(s,a)$ is the number of times rule a has been selected at s. The constant α, theoretically equal to $\sqrt{2}$, is used to balance exploration and exploitation. (2) *Expansion*. At the selected leaf node s, expand the search tree by applying all applicable production rules to generate its child nodes. (3) *Simulation*. For each child of s, conduct a fixed number of rollouts. In each rollout, generate a valid expression ϕ by randomly applying production rules. Then, estimate the optimal coefficients in ϕ and evaluate its fitness on the training data D. The reward for this rollout is typically defined as $1/(1+\texttt{NMSE}(\phi))$. (4) *Backpropagation*. Update the rewards and visit counts for node s and all its ancestors up to the root node. After the final iteration, the expression with the highest fitness score, encountered during rollouts, is returned as the best expression.

EGG*-based Backpropagation.* At the backpropagation step, we obtain the path from the root to the selected node s along with its estimated reward. Using EGG, we identify equivalent paths and update their estimated rewards simultaneously. This procedure effectively propagates information as if rollouts had been performed on all equivalent subtrees. Our update strategy is inspired by prior work, which uses a transposition table to explicitly store identical nodes in the search tree (Childs et al., 2008; Leurent & Maillard, 2020). The tables record all identical nodes through hashing-based methods and are effective in domains such as Go and Hearthstone. In symbolic regression, however, two nodes in the search tree can only be recognized as identical up to symbolic equivalence induced by rewrite rules, so a hashing-based transposition table cannot be directly applied.

The Backpropagation of EGG-MCTS is different from MCTS, but EGG-MCTS does not introduce bias. Indeed, the meaning of $\texttt{visits}(s)$ in Equation equation 2 changes: it no longer means times this specific tree node was on a simulated path, but times any representative of this equivalence class was visited. This is conceptually the same as a transposition table in MCTS that shares statistics across identical nodes, which accelerates learning without introducing bias.

Concretely, the path—representing a partially completed expression—is first converted into an initial e-graph. The e-graph is then saturated by repeatedly applying the set of rewrite rules. From this saturated e-graph, we sample several distinct equivalent sequences and check if the tree contains corresponding paths. If so, we apply backpropagation to all of them. In this way, equivalent expressions share the same reward information, used to avoid redundant search. Theorem 3.1 shows that EGG-MCTS accelerates learning by reducing the effective branching factor compared with standard MCTS, meaning fewer child nodes are explored at each learning iteration. The analysis is inspired from Czech et al. (2021), merging identical nodes in the tree as a single node in their directed graph.

Example 1. *An example execution pipeline of* EGG-*MCTS is provided in Figure 3. In Figure 3(d), we examine two distinct paths within the search tree maintained by the algorithm:*

$$\textit{Path } 1 : (A \to \log(A), A \to A \times A, A \to x_1), \qquad \textit{Node } 1 : s_1 = \log(x_1 \times A).$$
$$\textit{Path } 2 : (A \to A + A, A \to \log(A), A \to x_1, A \to \log(A)), \quad \textit{Node } 2 : s_2 = \log(x_1) + \log(A).$$

Here, each path is a sequence of edge labels from the root to the leaf node s_i. Based on the grammar definition in section 2, node s_1 corresponds to the partially completed expression $\log(x_1 \times A)$, while node s_2 represents $\log(x_1) + \log(A)$. The two nodes are equivalent under the rewrite rule $\log(ab) \rightsquigarrow \log a + \log b$. Consequently, their rewards, estimating the averaged goodness-of-fit of expressions on the dataset, should be approximately equal:

$$\texttt{Reward}(s_1, a) \approx \texttt{Reward}(s_2, a), \qquad \forall \textit{ rule } a \in \textit{ the set of available rules}$$

Exploring sub-trees of s_1 and s_2 independently leads to redundant computation, as MCTS repeatedly traverses equivalent regions of the search space, slowing down learning. With EGG-*MCTS, the visit counts and reward estimates of both paths are updated simultaneously, eliminating the need for additional rollouts on the green leaf in Figure 3(d) in future rounds. Please see Example 2 for the case of identity rewrite rules: $\sin^2(a) + \cos^2(a) \rightsquigarrow 1$ and $a/a \rightsquigarrow 1$.*

Embedding EGG into Deep Reinforcement Learning. DRL employs a neural network-based sequential decoder to predict a sequence of production rules, which are then converted into valid expressions according to the grammar. The reward function assigns high values to predicted expressions that fit the training data well. The model is trained to maximize the expected reward of generated expressions evaluated on the training data (Abolafia et al., 2018; Petersen et al., 2021; Landajuela et al., 2022; Jiang et al., 2024).

At each step, a neural network-based sequential decoder samples the next production rule from a probability distribution over all available rules, conditioned on the previously generated sequence. The model emits a sequence τ with probability $p_\theta(\tau)$. The reward function is typically defined as $\texttt{reward}(\tau) = 1/(1 + \texttt{NMSE}(\phi))$. The learning objective is to maximize the expected reward: $J(\theta) := \mathbb{E}_{\tau \sim p_\theta}[\texttt{reward}(\tau)]$, whose gradient is $\mathbb{E}_{\tau \sim p_\theta}[\texttt{reward}(\tau) \nabla_\theta \log p_\theta(\tau)]$. In practice, the expectation is approximated via Monte Carlo. Drawing N sequences $\{\tau_1, \ldots, \tau_N\}$ from the decoder, we obtain the policy gradient estimator:

$$g(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} (\texttt{reward}(\tau_i) - b) \nabla_\theta \log p_\theta(\tau_i), \tag{3}$$

where b is a baseline to reduce estimator variance (Weaver & Tao, 2001). Recently, Petersen et al. (2021) proposes to use the Top-25% quantile of samples to replace the classic empirical mean.

EGG-*based Policy Gradient Estimator.* For each sampled sequence τ_i, we construct an e-graph that compactly encodes all of its equivalent expressions. From this e-graph, we extract $K - 1$ equivalent sequences $\{\tau_i^{(2)}, \ldots, \tau_i^{(K)}\}$. The estimator is then revised to

$$g_{\texttt{egg}}(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left(\texttt{reward}(\tau_i) - b'\right) \nabla_\theta \log \left[\sum_{k=1}^{K} p_\theta(\tau_i^{(k)}) \right], \tag{4}$$

where $\tau_i^{(1)}$ is the original sequence τ_i and b' is the corresponding baseline, and $\sum_{k=1}^{K} p_\theta(\tau_i^{(k)})$ aggregates the probabilities of all equivalent sequences that share the same reward. The pipeline of DRL and EGG-DRL is presented in Figure 9. Theorem 3.2 shows that EGG-DRL achieves a lower-variance estimator compared to the estimator in standard DRL (Petersen et al., 2021).

Embed EGG into Large-Language Model. LLM is applied to search for optimal symbolic expressions with prompt tuning (Shojaee et al., 2025; Merler et al., 2024). The procedure consists of three key steps: (1) *Hypothesis Generation*: The LLM generates multiple candidate expressions based on a prompt describing the problem background and the definitions of each variable. (2) *Data-Driven Evaluation*: Each candidate expression is evaluated based on its fitness on the training dataset. (3) *Experience Management*: In subsequent iterations, the LLM receives feedback in the form of previously predicted expressions and their corresponding fitness scores, allowing it to refine future generations. High-fitness expressions are retained and updated over multiple rounds of iteration.

EGG-*based Feedback Prompt.* Since LLMs typically generate Python functions rather than symbolic expressions directly, we introduce a wrapper that parses the generated Python code into symbolic

expressions. These expressions are then transformed into e-graphs using a set of rewrite rules. From each e-graph, we extract semantically equivalent expressions and summarize them into a concise message, which is incorporated into the prompt for the next round. This augmentation enables the LLM to observe a richer set of functionally equivalent expressions, potentially improving the quality and accuracy of subsequent predictions.

3.3 Connection to Existing Methods

Sahoo et al. (2018); Li et al. (2024) propose representing expressions as layer-wise symbolic networks (SymNet), which are incompatible with our grammar definition. A follow-up work (Wu et al., 2024) on SymNet studies coefficient equivalence, showing that many coefficients obtained from SymNet can be merged into a single one. Systematically applying the e-graph framework to SymNet, to capture equivalence inside SymNet, remains an interesting open problem. For DRL-based approaches, there exist several extensions of the original method (Petersen et al., 2021), such as (Mundhenk et al., 2021b; Landajuela et al., 2022). It remains an interesting open problem to apply the symbolic equivalence to these extensions. Kamienny et al. (2022); Shojaee et al. (2023) propose to directly encode data into transformer and directly predict the traversal sequence of the expression. The end-to-end style model is trained using a cross-entropy loss. During inference, these models rely on pure sampling or planning-based approaches to select the best expressions from multiple samples. An intuitive way to integrate the EGG module into these frameworks is to use EGG to generate multiple correct output sequences during training. However, how to best exploit EGG during inference remains an open problem.

3.4 Theoretical Insight on Egg-SR Accelerating Learning

Theorem 3.1 establishes that augmenting MCTS with EGG yields an asymptotically tighter regret bound than standard MCTS, as the effective branching factor satisfies $\kappa_\infty \leq \kappa$. Also, Theorem 3.2 shows that embedding EGG into DRL produces an unbiased gradient estimator while strictly reducing gradient variance, ensuring more stable and efficient policy updates.

Theorem 3.1. *Consider the MCTS learning framework augmented with* EGG. *As defined in Definitions 1 and 3, let T denote the total number of learning iterations, $\gamma \in (0, 1)$ the discount factor of the corresponding Markov decision process, κ be the near-optimal branching factor of standard MCTS, and κ_∞ the corresponding branching factor of* EGG*-MCTS. Then the regret bounds satisfy*

$$\texttt{regret}(T) \;=\; \widetilde{\mathcal{O}}\left(n^{-\frac{\log(1/\gamma)}{\log \kappa}}\right) \qquad \texttt{regret}_{\texttt{egg}}(T) \;=\; \widetilde{\mathcal{O}}\left(n^{-\frac{\log(1/\gamma)}{\log \kappa_\infty}}\right) \qquad \textit{with } \kappa_\infty \leq \kappa$$

Proof Sketch. Our proof builds on part of the analysis of (Leurent & Maillard, 2020, Theorem 16), which employs a graph structure to replace the search tree to aggregate identical states. Their proof requires *unrolling* the graph into a tree that contains every path traversable in the graph. The search tree in our EGG-MCTS behaves almost identically to this unrolled tree. Following their regret analysis of the unrolled tree, we obtain the final regret bounds. A full proof is provided in Appendix B. □

Theorem 3.2. *Consider embedding* EGG *into the DRL framework. Let N samples be drawn from the distribution p_θ, and* EGG *module provides additional $K-1$ equivalent variants for each sample. (1) Unbiasedness. The expectation of the standard estimator $g(\theta)$ equals that of the* EGG*-based estimator $g_{\texttt{egg}}(\theta)$: $\mathbb{E}_{\tau\sim p_\theta}\left[g(\theta)\right] = \mathbb{E}_{\tau\sim p_\theta}\left[g_{\texttt{egg}}(\theta)\right]$. (2) Variance Reduction. The variance of the proposed estimator is smaller than that of the standard estimator:* $\mathrm{Var}_{\tau\sim p_\theta}\left[g_{\texttt{egg}}(\theta)\right] \leq \mathrm{Var}_{\tau\sim p_\theta}\left[g(\theta)\right]$.

Proof Sketch. To show unbiasedness, we follow the definition of the estimators to verify that the expected value of $g_{\texttt{egg}}(\theta)$ equals that of $g(\theta)$. The key idea behind the variance reduction is that EGG groups together equivalent sequences that share the same reward. Grouping sequences with identical rewards leads to lower within-group variance. Full proof is provided in Appendix C. □

4 Related Works

Symbolic Equivalence is a core concept in program synthesis and mathematical reasoning (Willsey et al., 2021). In SQL query optimization, it enables rewriting queries into equivalent but more efficient forms (Barbulescu et al., 2024). In hardware synthesis, it supports cost-aware rewrites such

as optimized matrix multiplications (Ustun et al., 2022). In formal methods, it accelerates automated theorem proving via normalization and equivalence checking (Kurashige et al., 2024). In mathematical language processing, it enables the generation of paraphrased forms of mathematical expressions (Zheng et al., 2025). In symbolic regression, de França & Kronberger (2023) leverages e-graphs to mitigate over-parameterization of candidate expressions. de França & Kronberger (2025); de Franca & Kronberger (2025) incorporates e-graphs into genetic programming (GP) to detect and eliminate redundant individuals, encouraging the GP method to explore novel expression. A recent follow-up work rEGGression (de França & Kronberger, 2025) provides a richer API for interacting with the GP method. Further, our approach advances this idea by providing a simple, unified interface for encoding known mathematical equalities as e-graphs, thereby enabling symbolic-equivalence-aware learning in a wide range of symbolic regression algorithms, together with provable theoretical guarantees.

Knowledge-Guided Scientific Discovery. Recent efforts have explored incorporating physical and domain-specific knowledge to accelerate the symbolic discovery process. AI-Feynman (Udrescu & Tegmark, 2020; Udrescu et al., 2020; Keren et al., 2023; Cornelio et al., 2023) constrained the search space to expressions that exhibit compositionality, additivity, and generalized symmetry. Similarly, Tenachi et al. (2023) encoded physical unit constraints into learning to eliminate physically impossible solutions. Other works further constrained the search space by integrating user-specified hypotheses and prior knowledge, offering a guided approach to symbolic regression (Bendinelli et al., 2023; Kamienny, 2023; Shojaee et al., 2025; Taskin et al., 2025). Zhang et al. (2025) propose using retrieval-augmented generation to acquire new, informative data that accelerates the overall learning process. Our EGG presents a new idea in knowledge-guided learning that is orthogonal to existing approaches. An interesting direction for future work is to investigate whether EGG can be combined with these knowledge-guided strategies to further improve symbolic discovery.

Equivalence-aware Learning. Equivalence and symmetry have long been recognized as crucial for improving efficiency in search and learning. In MCTS, transposition tables exploit graph equivalence by merging nodes that represent the same underlying state, avoiding redundant rollouts and accelerating convergence (Childs et al., 2008). More recent extensions explicitly leverage symmetries to prune symmetric branches of the search space (Saffidine et al., 2012; Leurent & Maillard, 2020). In reinforcement learning, symmetries in the state–action space have been used to accelerate convergence (Grimm et al., 2020). Large Language Models also benefit from equivalence-awareness, particularly in code generation (Sharma & David, 2025).

5 EXPERIMENTS

We show that (1) EGG enhances existing learning algorithms in discovering expressions with smaller Normalized MSEs. (2) In case studies, EGG consistently exhibits both time and space efficiency. The exact experiment configuration for the following results and analysis is in Appendix E.

5.1 OVERALL BENCHMARKS

Impact of EGG on MCTS.

We conduct two analyses to evaluate the impact of integrating EGG into standard MCTS: (1) the median normalized MSE of the TopK ($K = 10$) expressions identified at the end of training, and (2) the growth of the search tree, measured by the number of explored nodes over learning iterations. The dataset is selected from Jiang & Xue (2023) as the expressions contain $\sin, \cos$ operators, which contain many symbolic-equivalence variants. The detailed experimental setups, including the rewrite rules and datasets used in each comparison, are provided in Appendix E. Table 1 shows that

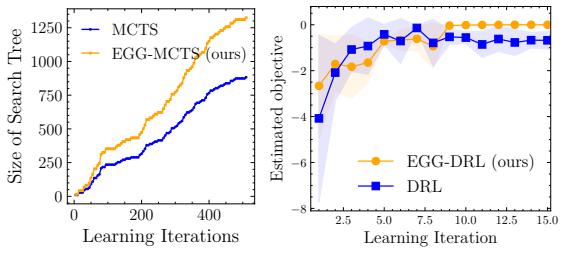


Figure 4: On the "sincos(3,2,2)" dataset, we show **(Left)** Search tree size over learning iterations for MCTS and EGG-MCTS, and also **(Right)** Empirical mean and standard deviation of the estimated quantity for DRL and EGG-DRL.

Table 1: On Trigonometric datasets, median NMSE values of the best-predicted expressions found by all the algorithms. The 3-tuples at the top $(\cdot, \cdot, \cdot)$ indicate the number of free variables, singular terms, and cross terms in the ground-truth expressions generating the dataset. The set of operators is $\{\sin, \cos, +, -, \times\}$. The best result in each column is underlined.

	Noiseless Setting				**Noisy Setting**			
	$(2,1,1)$	$(3,2,2)$	$(4,4,6)$	$(5,5,5)$	$(2,1,1)$	$(3,2,2)$	$(4,4,6)$	$(5,5,5)$
EGG-MTCS	<1E-6	<1E-6	0.006	0.009	0.005	0.012	0.091	0.121
MTCS	0.006	0.033	0.144	0.147	0.015	0.007	0.138	0.150
EGG-DRL	0.020	0.161	2.381	2.168	0.07	0.35	5.09	5.67
DRL	0.030	0.277	2.990	2.903	0.09	0.44	2.46	14.44

Table 2: Comparison of LLM, and EGG-LLM models on different scientific benchmark problems measured by the NMSE metric. The best result in each column is underlined.

	Oscillation I		**Oscillation II**		**Bacterial growth**		**Stress-Strain**	
	IID↓	OOD↓	IID↓	OOD↓	IID↓	OOD↓	IID↓	OOD↓
EGG-LLM (GPT3.5)	<1E-6	0.0004	<1E-6	<1E-6	0.0121	0.0198	0.0202	0.0419
LLM-SR (GPT-3.5)	<1E-6	0.0005	<1E-6	3.81E-5	0.0214	0.0264	0.0210	0.0516
EGG-LLM (Mistral)	<1E-6	0.0002	0.0021	0.0114	0.0101	0.0107	0.0133	0.0754
LLM-SR (Mistral)	<1E-6	0.0002	0.0030	0.0291	0.0026	0.0037	0.0162	0.0946

EGG-MCTS consistently discovers expressions with lower normalized quantile scores compared to standard MCTS. Figure 4(Left) illustrates that EGG-MCTS maintains a broader and deeper search tree, indicating exploration of a larger and more diverse search space. Across various datasets, augmenting MCTS with EGG improves symbolic expression accuracy. This improvement is primarily due to the effectiveness of our rewrite rules, which cover a rich set of trigonometric identities and enable efficient exploration of symbolic variants in trigonometric expression spaces.

Impact of EGG on DRL. Table 1 reports the median NMSE values of the best-predicted expressions discovered by EGG-DRL and standard DRL, under identical experiment settings. Expressions returned by EGG-DRL achieve a smaller NMSE value on noiseless and noisy settings. It shows that embedding EGG into DRL helps to discover expressions with better NMSE. In Figure 4 (Right), we plot the estimated objective, defined as $R(\tau_i) \log p_\theta(\tau_i)$ where each trajectory τ_i is sampled from the sequential decoder with probability $p_\theta(\tau_i)$ (see Equation 3). We plot the empirical mean and standard deviation of this objective over training iterations. The observed reduction in variance is primarily due to the symbolic variants generated via the e-graph, which enable averaging over multiple equivalent expressions and thus yield more stable gradients.

Impact of EGG on LLM. We evaluate the impact of augmenting LLMs with richer feedback prompts that incorporate equivalent expressions generated by EGG. Following the dataset and experimental setup from the original paper (Shojaee et al., 2025), we summarize the results in Table 2. The result of LLM-SR directly uses the reported result in Shojaee et al. (2025). The results show that integrating EGG enables the LLM to discover higher-quality expressions under the same experimental conditions.

5.2 CASE ANALYSIS

Space Efficiency of EGG. We evaluate the space efficiency of the e-graph representation in comparison to a traditional array-based approach. The array-based method explicitly stores each expression variant as a unique sequence, leading to exponential memory growth. In contrast, the e-graph compactly encodes multiple equivalent expressions by sharing common sub-expressions.

We benchmark the memory consumption of storing all equivalent variants of input expressions under two settings: (1) $\phi = \log(x_1 \times \ldots \times x_n)$, using the logarithmic identity $\log(ab) \rightsquigarrow \log a + \log b$, and (2) $\phi = \sin(x_1 + \ldots + x_n)$, using the trigonometric identity $\sin(a + b) \rightsquigarrow \sin(a)\cos(b) + \sin(b)\cos(a)$. Both settings yield 2^{n-1} equivalent variants.

Figure 5 presents the memory consumption as a function of the input variable size n. The results show that the e-graph representation achieves significantly lower memory usage compared to the array-based approach. Figure 5(right) visualizes two sample e-graphs, which illustrate how shared

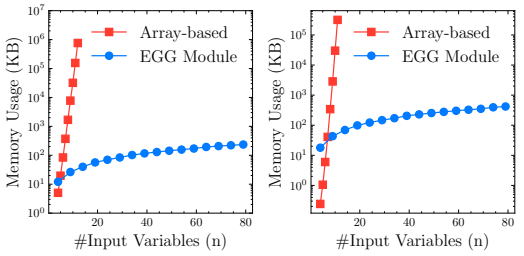


Figure 5: EGG uses less memory than the array-based approach for two settings: **(Left)** $\log(x_1 \times \dots x_n)$ rewritten using $\log(ab) \rightsquigarrow \log a + \log b$. **(Right)** $\sin(x_1 + \dots x_n)$ rewritten using $\sin(a+b) \rightsquigarrow \sin a \cos b + \sin a \cos b$.

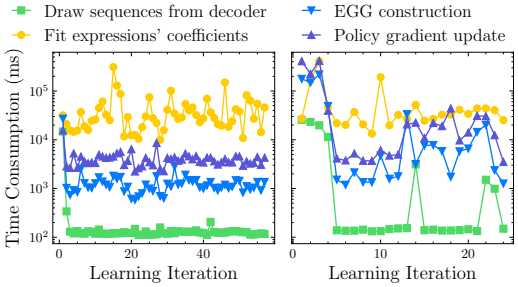


Figure 6: The EGG module is time efficient and introduces negligible time overhead, compared with four main computations in DRL. **Left:** LSTM. **Right:** Decoder-only Transformer.

sub-expressions are stored only once, contributing to the space efficiency of EGG. The corresponding e-graphs for $n = 2, 3, 4$ are visualized in Appendix Figures 16 and 17.

Time Efficiency of EGG with DRL. As shown in Figure 6, we benchmark the runtime of the four main computations in EGG-DRL on the selected "sincos(3,2,2)" dataset: (1) sampling sequences of rules from the sequential decoder, (2) fitting coefficients in symbolic expressions to the training data, (3) generating equivalent expressions via EGG, and (4) computing the loss, gradients, and updating the neural network parameters. We consider two settings for the sequential decoder: a 3-layer LSTM with hidden dimension 128, and a decoder-only Transformer with 6 attention heads and hidden dimension 128. The EGG module contributes minimal computational overhead relative to more expensive steps such as coefficient fitting and network updates. This highlights the practicality of incorporating EGG into DRL-based symbolic regression frameworks.

Additional E-graph Visualizations. To further demonstrate the effectiveness of the proposed EGG module, we present several additional e-graph visualizations generated using our implemented API on selected complex expressions from the Feynman dataset (Udrescu & Tegmark, 2020), each highlighting different rewrite rules (see Appendix F.2).

6 CONCLUSION

In this paper, we introduced EGG-SR, a unified framework that integrates symbolic equivalence into symbolic regression through equality graphs (e-graphs) to accelerate the discovery of optimal expressions. Our theoretical analysis establishes the advantages of EGG-MCTS over standard MCTS and EGG-DRL over conventional DRL algorithms. Extensive experiments further demonstrate that EGG consistently enhances the ability of existing methods to uncover high-quality governing equations from experimental data. Currently, the collection of rewrite rules relies on manual effort. In this work, the set of rewrite rules is manually chosen for the dataset. In future work, we plan to develop adaptive approaches for discovering more rewrite rules during the learning iterations and to extend the proposed method to related tasks, such as code optimization and program synthesis.

Ethics Statement. All authors have read and commit to adhering to the ICLR Code of Ethics. This work uses only publicly available datasets and open-source models; no human subjects or sensitive personal data are involved, and Institutional Review Board approval is not required. We confirm that the work does not target discriminatory or harmful applications. Any potential conflicts of interest have been disclosed. We do not foresee misuse beyond the general risks associated with large language models, and we report methods and results transparently.

Reproducibility Statement. To facilitate reproduction, the main text and Appendix describe the model architecture, training procedures, and evaluation protocols, and the Appendix includes complete proofs of theoretical results. All datasets are publicly available, and data preprocessing steps are documented in the submission. We include training and evaluation code and brief instructions in the supplementary material and refer to the relevant sections rather than repeating details here.

REFERENCES

Daniel A. Abolafia, Mohammad Norouzi, and Quoc V. Le. Neural program synthesis with priority queue training. *CoRR*, abs/1801.03526, 2018.

Miltiadis Allamanis, Pankajan Chanthirasegaran, Pushmeet Kohli, and Charles Sutton. Learning continuous semantic representations of symbolic expressions. In *ICML*, volume 70 of *Proceedings of Machine Learning Research*, pp. 80–88. PMLR, 2017.

George-Octavian Barbulescu, Taiyi Wang, Zak Singh, and Eiko Yoneki. Learned graph rewriting with equality saturation: A new paradigm in relational query rewrite and beyond. *CoRR*, abs/2407.12794, 2024.

Zachary Bastiani, Robert M Kirby, Jacob Hochhalter, and Shandian Zhe. Diffusion-based symbolic regression. *arXiv preprint arXiv:2505.24776*, 2025.

Tommaso Bendinelli, Luca Biggio, and Pierre-Alexandre Kamienny. Controllable neural symbolic regression. In *ICML*, volume 202 of *Proceedings of Machine Learning Research*, pp. 2063–2077. PMLR, 2023.

Jure Brence, Ljupco Todorovski, and Saso Dzeroski. Probabilistic grammars for equation discovery. *Knowl. Based Syst.*, 224:107077, 2021.

George Casella and Christian P. Robert. Rao-blackwellisation of sampling schemes. *Biometrika*, 83 (1):81–94, 1996.

William G. La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H. Moore. Contemporary symbolic regression methods and their relative performance. In *NeurIPS Datasets and Benchmarks*, 2021.

Benjamin E. Childs, James H. Brodeur, and Levente Kocsis. Transpositions and move groups in monte carlo tree search. In *CIG*, pp. 389–395. IEEE, 2008.

Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Cristina Cornelio, Sanjeeb Dash, Vernon Austel, Tyler R Josephson, Joao Goncalves, Kenneth L Clarkson, Nimrod Megiddo, Bachir El Khadir, and Lior Horesh. Combining data and theory for derivable scientific discovery with ai-descartes. *Nature Communications*, 14(1):1777, 2023.

Ryan Cory-Wright, Cristina Cornelio, Sanjeeb Dash, Bachir El Khadir, and Lior Horesh. Evolving scientific discovery by unifying data and background knowledge with ai hilbert. *Nature Communications*, 15(1):5922, 2024.

Johannes Czech, Patrick Korus, and Kristian Kersting. Improving alphazero using monte-carlo graph search. In *ICAPS*, pp. 103–111. AAAI Press, 2021.

Fabrício Olivetti de França and Gabriel Kronberger. Reducing overparameterization of symbolic regression models with equality saturation. In *GECCO*, pp. 1064–1072. ACM, 2023.

Fabrício Olivetti de França and Gabriel Kronberger. Improving genetic programming for symbolic regression with equality graphs. In *GECCO*. ACM, 2025.

Fabricio Olivetti de Franca and Gabriel Kronberger. Equality graph assisted symbolic regression. *arXiv preprint arXiv:2511.01009*, 2025.

Fabricio Olivetti de França and Gabriel Kronberger. reggression: an interactive and agnostic tool for the exploration of symbolic regression models. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '25, pp. 4–12, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400714658.

Wei Deng, Qi Feng, Georgios Karagiannis, Guang Lin, and Faming Liang. Accelerating convergence of replica exchange stochastic gradient MCMC via variance reduction. In *ICLR*. OpenReview.net, 2021.

Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2000.

Steven Ganzert, Josef Guttmann, Daniel Steinmann, and Stefan Kramer. Equation discovery for model identification in respiratory mechanics of the mechanically ventilated human lung. In *Discovery Science*, volume 6332, pp. 296–310. Springer, 2010.

Amir Kafshdar Goharshady, Chun Kit Lam, and Lionel Parreaux. Fast and optimal extraction for sparse equality graphs. *Proc. ACM Program. Lang.*, 8(OOPSLA2):2551–2577, 2024.

Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10): 2222–2232, 2016.

Christopher Grimm, André Barreto, Satinder Singh, and David Silver. The value equivalence principle for model-based reinforcement learning. *NeurIPS*, 33:5541–5552, 2020.

Gérard Huet and Derek C Oppen. Equations and rewrite rules: A survey. *Formal Language Theory*, pp. 349–405, 1980.

Nan Jiang and Yexiang Xue. Symbolic regression via control variable genetic programming. In *ECML/PKDD*, pp. 178–195. Springer, 2023.

Nan Jiang, Md. Nasim, and Yexiang Xue. Vertical symbolic regression via deep policy gradient. In *IJCAI*, pp. 5891–5899. ijcai.org, 2024.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. *Advances in neural information processing systems*, 26, 2013.

Paul Kahlmeyer, Joachim Giesen, Michael Habeck, and Henrik Voigt. Scaling up unbiased search-based symbolic regression. In Kate Larson (ed.), *IJCAI-24*, pp. 4264–4272. International Joint Conferences on Artificial Intelligence Organization, 8 2024.

Pierre-Alexandre Kamienny. *Efficient adaptation of reinforcement learning agents: from model-free exploration to symbolic world models*. Theses, Sorbonne Université, October 2023.

Pierre-Alexandre Kamienny, Stéphane d'Ascoli, Guillaume Lample, and François Charton. End-to-end symbolic regression with transformers. In *NeurIPS*, 2022.

Pierre-Alexandre Kamienny, Guillaume Lample, Sylvain Lamprier, and Marco Virgolin. Deep generative symbolic regression with monte-carlo-tree-search. In *ICML*, volume 202. PMLR, 2023.

Liron Simon Keren, Alex Liberzon, and Teddy Lazebnik. A computational framework for physics-informed symbolic regression with straightforward integration of domain knowledge. *Scientific Reports*, 13(1):1249, 2023.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *ECML*, volume 4212 of *Lecture Notes in Computer Science*, pp. 282–293. Springer, 2006.

Cole Kurashige, Ruyi Ji, Aditya Giridharan, Mark Barbone, Daniel Noor, Shachar Itzhaky, Ranjit Jhala, and Nadia Polikarpova. Cclemma: E-graph guided lemma discovery for inductive equational proofs. *Proc. ACM Program. Lang.*, 8(ICFP):818–844, 2024.

William La Cava, Patryk Orzechowski, Bogdan Burlacu, Fabrício Olivetti de França, Marco Virgolin, Ying Jin, Michael Kommenda, and Jason H Moore. Contemporary symbolic regression methods and their relative performance. *arXiv preprint arXiv:2107.14351*, 2021.

Kyle J. LaFollette, Janni Yuval, Roey Schurr, David Melnikoff, and Amit Goldenberg. Data-driven equation discovery reveals nonlinear reinforcement learning in humans. *Proceedings of the National Academy of Sciences*, 122(31):e2413441122, 2025.

Mikel Landajuela, Chak Shing Lee, Jiachen Yang, Ruben Glatt, Cláudio P. Santiago, Ignacio Aravena, Terrell Nathan Mundhenk, Garrett Mulcahy, and Brenden K. Petersen. A unified framework for deep symbolic regression. In *NeurIPS*, 2022.

Edouard Leurent and Odalric-Ambrym Maillard. Monte-carlo graph search: the value of merging similar states. In *ACML*, volume 129 of *Proceedings of Machine Learning Research*, pp. 577–592. PMLR, 2020.

Wenqiang Li, Weijun Li, Lina Yu, Min Wu, Linjun Sun, Jingyi Liu, Yanjie Li, Shu Wei, Yusong Deng, and Meilan Hao. A neural-guided dynamic symbolic network for exploring mathematical expressions from data. In *ICML*. OpenReview.net, 2024.

He Ma, Arunachalam Narayanaswamy, Patrick Riley, and Li Li. Evolving symbolic density functionals. *Science Advances*, 8(36), 2022.

Matteo Merler, Katsiaryna Haitsiukevich, Nicola Dainese, and Pekka Marttinen. In-context symbolic regression: Leveraging large language models for function discovery. In *ACL (Student Research Workshop)*, pp. 589–606. Association for Computational Linguistics, 2024.

T. Nathan Mundhenk, Mikel Landajuela, Ruben Glatt, Cláudio P. Santiago, Daniel M. Faissol, and Brenden K. Petersen. Symbolic regression via deep reinforcement learning enhanced genetic programming seeding. In *NeurIPS*, pp. 24912–24923, 2021a.

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. In *NeurIPS*, volume 34, pp. 24912–24923, 2021b.

Rémi Munos. From bandits to monte-carlo tree search: The optimistic principle applied to optimization and planning. *Found. Trends Mach. Learn.*, 7(1):1–129, 2014.

Nico JD Nagelkerke et al. A note on a general definition of the coefficient of determination. *Biometrika*, 78(3):691–692, 1991.

Chandrakana Nandi, Max Willsey, Amy Zhu, Yisu Remy Wang, Brett Saiki, Adam Anderson, Adriana Schulz, Dan Grossman, and Zachary Tatlock. Rewrite rule inference using equality saturation. *Proc. ACM Program. Lang.*, 5(OOPSLA):1–28, 2021.

Matteo Papini, Damiano Binaghi, Giuseppe Canonaco, Matteo Pirotta, and Marcello Restelli. Stochastic variance-reduced policy gradient. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4023–4032. PMLR, 2018.

Brenden K. Petersen, Mikel Landajuela, T. Nathan Mundhenk, Cláudio Prata Santiago, Sookyung Kim, and Joanne Taery Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *ICLR*, 2021.

Abdallah Saffidine, Tristan Cazenave, and Jean Méhat. UCD : Upper confidence bound for rooted directed acyclic graphs. *Knowl. Based Syst.*, 34:26–33, 2012.

Subham S. Sahoo, Christoph H. Lampert, and Georg Martius. Learning equations for extrapolation and control. In *ICML*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4439–4447. PMLR, 2018.

Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent advances in recurrent neural networks. *arXiv preprint arXiv:1801.01078*, 2017.

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

Saul Shanabrook. Egglog python: A pythonic library for e-graphs, 2024.

Arindam Sharma and Cristina David. Assessing correctness in llm-based code generation via uncertainty estimation. *arXiv preprint arXiv:2502.11620*, 2025.

Parshin Shojaee, Kazem Meidani, Amir Barati Farimani, and Chandan K. Reddy. Transformer-based planning for symbolic regression. In *NeurIPS*, 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 *ICLR*, 2025.

Fangzheng Sun, Yang Liu, Jian-Xun Wang, and Hao Sun. Symbolic physics learner: Discovering governing equations via monte carlo tree search. In *ICLR*, 2023.

Bilge Taskin, Wenxiong Xie, and Teddy Lazebnik. Knowledge integration for physics-informed symbolic regression using pre-trained large language models. *arXiv preprint arXiv:2509.03036*, 2025.

Wassim Tenachi, Rodrigo Ibata, and Foivos I. Diakogiannis. Deep symbolic regression for physics guided by units constraints: toward the automated discovery of physical laws. *CoRR*, abs/2303.03192, 2023.

Ljupco Todorovski and Saso Dzeroski. Declarative bias in equation discovery. In *ICML*, pp. 376–384. Morgan Kaufmann, 1997.

Silviu-Marian Udrescu and Max Tegmark. Ai feynman: A physics-inspired method for symbolic regression. *Science Advances*, 6(16):eaay2631, 2020. doi: 10.1126/sciadv.aay2631.

Silviu-Marian Udrescu, Andrew K. Tan, Jiahai Feng, Orisvaldo Neto, Tailin Wu, and Max Tegmark. AI feynman 2.0: Pareto-optimal symbolic regression exploiting graph modularity. In *NeurIPS*, 2020.

Ecenur Ustun, Ismail San, Jiaqi Yin, Cunxi Yu, and Zhiru Zhang. Impress: Large integer multiplication expression rewriting for FPGA HLS. In *FCCM*, pp. 1–10. IEEE, 2022.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *NeurIPS*, 30:5998–6008, 2017.

Marco Virgolin and Solon P Pissis. Symbolic regression is NP-hard. *TMLR*, 2022.

Uwe Waldmann, Sophie Tourret, Simon Robillard, and Jasmin Blanchette. A comprehensive framework for saturation theorem proving. *J. Autom. Reason.*, 66(4):499–539, 2022.

Lex Weaver and Nigel Tao. The optimal reward baseline for gradient-based reinforcement learning. In *UAI*, pp. 538–545. Morgan Kaufmann, 2001.

Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. egg: Fast and extensible equality saturation. *Proc. ACM Program. Lang.*, 5(POPL): 1–29, 2021.

Min Wu, Weijun Li, Lina Yu, Wenqiang Li, Jingyi Liu, Yanjie Li, and Meilan Hao. Prunesymnet: A symbolic neural network and pruning algorithm for symbolic regression. *CoRR*, abs/2401.15103, 2024.

Ziyu Xiang, Kenna Ashen, Xiaofeng Qian, and Xiaoning Qian. Graph-based symbolic regression with invariance and constraint encoding. In *NeurIPS*, 2025.

Hengzhe Zhang, Qi Chen, Bing Xue, Wolfgang Banzhaf, and Mengjie Zhang. RAG-SR: retrieval-augmented generation for neural symbolic regression. In *ICLR*. OpenReview.net, 2025.

Hongbo Zheng, Suyuan Wang, Neeraj Gangwar, and Nickvash Kani. E-gen: Leveraging e-graphs to improve continuous representations of symbolic expressions. In *NAACL (Long Papers)*, pp. 11772–11788. Association for Computational Linguistics, 2025.

Zeyu Zheng, Junhyuk Oh, and Satinder Singh. On learning intrinsic rewards for policy gradient methods. In *NeurIPS*, pp. 4649–4659, 2018.

Contents

Use of Large Language Models (LLMs). A large language model (LLM) was used exclusively for language refinement, including improvements in grammar, clarity, and readability. All research ideas, methodology, analyses, and conclusions are entirely the authors' own. The authors have thoroughly reviewed the final manuscript and accept full responsibility for its content.

Availability of EGG, Baselines and Dataset. Please find our code repository at:

`https://anonymous.4open.science/r/egg-sr`

- **EGG**. The implementation of our EGG module is located in the folder `src/equality_graph/`.
- **Datasets**. The list of datasets used in our experiments can be found in `datasets/scibench/scibench/data/`.
- **Baselines**. The implementations of several baseline algorithms, along with our adapted versions, are provided in the `algorithms/` folder. Execution scripts for running each algorithm are also included.

We also provide a `README.md` file with instructions for installing the required Python packages and dependencies.

We summarize the supplementary material as follows: Section A provides implementation details of the proposed EGG module. Sections B and C present a detailed theoretical analysis of the proposed method. Section E describes the experimental setup and configurations. Finally, Section F presents additional experimental results.

A IMPLEMENTATION OF EGG MODULE

In what follows, we present the implementation of a unified framework that employs a context-free grammar to represent symbolic expressions, rewrite rules, and e-graphs.

A.1 CONTEXT-FREE GRAMMAR DEFINITION FOR SYMBOLIC EXPRESSION

The context-free grammar for symbolic regression is defined in Section 2. Figure 7 further illustrates how an expression such as $\phi = c_1 \log(x_1)$ can be constructed from the start symbol by sequentially applying grammar rules. We begin with the multiplication rule $A \to A \times A$, which expands the initial symbol A in $\phi = A$ to yield $\phi = A \times A$. Continuing this process, each non-terminal symbol is recursively expanded using the appropriate grammar rules until we obtain the valid expression $\phi = c_1 \log(x_1)$.

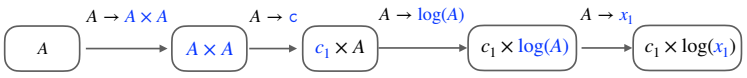


Figure 7: Transforming a sequence of grammar rules into a valid expression. At each step, the *first* non-terminal symbol inside the squared box is expanded. The expanded parts are color-highlighted.

The next step is to determine the optimal coefficient c_1 in the expression $\phi = c_1 \log(x_1)$. More generally, suppose the expression contains m free coefficients. Given training data $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, we optimize these coefficients using a gradient-based method such as BFGS by solving

$$\mathbf{c}^* \leftarrow \arg\min_{\mathbf{c}\in\mathbb{R}^m} \frac{1}{N} \sum_{i=1}^{N} \ell(\phi(\mathbf{x}_i, \mathbf{c}), y_i),$$

where the loss function ℓ is typically the normalized mean squared error (NMSE), which is defined in Equation 12.

A.2 Implementation of Rewrite Rules

The following snippet defines the Python implementation of a rewrite rule from a known mathematical identity. The function `rules_to_s_expr` converts the input list representation into an internal symbolic expression tree:

```
class Rule:
    def __init__(self, lhs, rhs):
        self.lhs = rules_to_s_expr(lhs)
        self.rhs = rules_to_s_expr(rhs)
```

Each rule encodes one direction of applying a mathematical identity. Prior research (de França & Kronberger, 2025) has primarily used rewrite rules to simplify expressions, typically rewriting a longer expression into a shorter one. In contrast, our work considers *both* directions of each identity, enabling us to systematically generate the complete set of symbolically equivalent expressions.

Table 3: The list of rewrite rules for trigonometric functions, each derived from a known trigonometric law.

Category	rewrite rules
Sum-to-Product Identities	$\sin(a)+\sin(b) \rightsquigarrow 2\sin\left(\frac{a+b}{2}\right)\cos\left(\frac{a-b}{2}\right)$ $\cos(a)+\cos(b) \rightsquigarrow 2\cos\left(\frac{a+b}{2}\right)\cos\left(\frac{a-b}{2}\right)$
Product-to-Sum Identities	$\sin(a)\cos(b) \rightsquigarrow \frac{1}{2}[\sin(a+b)+\sin(a-b)]$ $\cos(a)\cos(b) \rightsquigarrow \frac{1}{2}[\cos(a+b)+\cos(a-b)]$ $\sin(a)\sin(b) \rightsquigarrow \frac{1}{2}[\cos(a-b)-\cos(a+b)]$
Double Angle Formulas	$\sin(a+a) \rightsquigarrow 2\sin(a)\cos(a)$ $\cos(a+a) \rightsquigarrow \cos^2(a)-\sin^2(a)$ $\cos(a+a) \rightsquigarrow 2\cos^2(a)-1$ $\cos(a+a) \rightsquigarrow 1-2\sin^2(a)$ $\tan(a+a) \rightsquigarrow \frac{2\tan(a)}{1-\tan^2(a)}$
Pythagorean Identities	$\sin^2(a)+\cos^2(a) \rightsquigarrow 1$ $\sec^2(a)-\tan^2(a) \rightsquigarrow 1$ $\csc^2(a)-\cot^2(a) \rightsquigarrow 1$
Half-Angle Formulas	$\sin^2\left(\frac{a}{2}\right) \rightsquigarrow \frac{1-\cos(a)}{2}$ $\cos^2\left(\frac{a}{2}\right) \rightsquigarrow \frac{1+\cos(a)}{2}$ $\tan^2\left(\frac{a}{2}\right) \rightsquigarrow \frac{1-\cos(a)}{1+\cos(x)}.$
Sum and Difference Formulas	$\sin(a\pm b) \rightsquigarrow \sin(a)\cos(b)\pm\cos(a)\sin(b)$ $\cos(a\pm b) \rightsquigarrow \cos(a)\cos(b)\mp\sin(a)\sin(b)$ $\tan(a\pm b) \rightsquigarrow (\tan(a)\pm\tan(b))\,/\,(1\mp\tan(a)\tan(b))$

We define several types of rewrite rules based on well-known mathematical identities: Let a, b denote arbitrary variables, coefficients, or sub-expressions.

1. Commutative law. $a+b=b+a$ or $a*b=b*a$ will be converted into rules: $a+b \rightsquigarrow b+a, a*b \rightsquigarrow b*a$.

```
# Commutative laws
Rule(lhs=['A->(A+A)', 'A->a', 'A->b'], rhs=['A->(A+A)', 'A->b', 'A->a'])
Rule(lhs=['A->A*A', 'A->a', 'A->b'], rhs=['A->A*A', 'A->b', 'A->a'])
```

2. Distributive laws: $(a+b)^2 \rightsquigarrow a^2+2ab+b^2$.
3. Factorization. Expressions can be decomposed into simpler components: $a^2-b^2 \rightsquigarrow (a-b)(a+b)$.
4. Exponential and logarithmic identities: $\exp(a+b) \rightsquigarrow \exp(a)\times\exp(b)$, $\log(ab) \rightsquigarrow \log(a)+\log(b)$, and also $\log(a^b) \rightsquigarrow b\log a$,
5. The rewrite rules for trigonometric identities are summarized in Table 3. For example, The rule $\sin(a+b) \rightsquigarrow \sin(a)\cos(b)+\sin(b)\cos(a)$ is implemented as follow:

```
Rule(
lhs=['A->sin(A)','A->(A+A)', 'A->a', 'A->b'],
rhs=['A->(A+A)', 'A->A*A', 'A->sin(A)','A->a','A->cos(A)', 'A->b',
                           'A->cos(A)','A->a','A->sin(A)', 'A->b'])
```

A visualization of this rule being applied to an expression is shown in Figure 14.

For practical reasons, we did not introduce rewrite rules that could make the e-graph exponentially large. As a result, a large family of common rules (e.g., $a + b - b \rightsquigarrow a - b + b$) is omitted. Such extreme cases are still beyond the capability of the current EGG module.

A.3 IMPLEMENTATION OF E-GRAPH FUNCTIONALITIES

Here we provide the implementation of the EGG data structure proposed in Section 3.1. An e-graph consists of two core components: *e-nodes* and *e-classes*. The following snippet presents the skeleton structure of these two components:

```python
class ENode:
    def __init__(self, operator: str, operands: List):
        self.operator, self.operands = operator, operands

    def canonicalize(self):
        # Convert to the canonical form of this ENode

class EClassID:
    def __init__(self, id):
        self.id = id
        self.parent = None
    def find_parent(self):
        # Return the representative parent ID (using union-find algorithm)
    def __repr__(self):
        return 'e-class{}'.format(self.id)
```

The e-graph structure is implemented in the following class. The dictionary `hashcons` stores all the edges from an e-node to an e-class. The API supports the key operations of (1) adding new expressions to the e-graph by calling `add_enode`, and (2) merging two e-classes into one:

```python
class EGraph:
    # hashcons: stores a mapping from ENode to its corresponding EClass
    hashcons: Dict[ENode, EClassID] = {}

    def add_enode(self, enode: ENode):
        # Add an ENode into the e-graph

    def eclasses(self) -> Dict[EClassID, List[ENode]]:
        # Return all e-classes and their associated e-nodes

    def merge(self, a: EClassID, b: EClassID):
        # Merge two e-classes using the union-find structure

    def substitute(self, pattern: Node, ...) -> EClassID:
        # Construct a new expression in the e-graph from a rule RHS
```

Construction The e-graph is built by repeatedly applying a set of rewrite rules to the e-graph. Each rewrite rule consists of a left-hand side (LHS) and a right-hand side (RHS) expression. The function `apply_rewrite_rules` determines which new subgraphs to construct and where to merge them within the existing e-graph.

Specifically, the LHS pattern of each rule is matched against the e-graph using the `ematch` function, which identifies all subexpressions that match the pattern. For each match, the RHS is instantiated using the extracted variable bindings, and the resulting expression is inserted into the

e-graph and merged with the corresponding e-class. The `equality_saturation` function invokes `apply_rewrite_rules` for a fixed number of iterations until saturation is reached.

```
def apply_rewrite_rules(eg: EGraph, rules: List[Rule]):
    eclasses = eg.eclasses()
    matches = []
    for rule in rules:
        for eid, env in ematch(eclasses, rule.lhs):
            matches.append((rule, eid, env))
    for rule, eid, env in matches:
        new_eid = eg.substitute(rule.rhs, env)
        eg.merge(eid, new_eid)
    eg.rebuild()

def equality_saturation(fn, rules, max_iter=20):
    eg = EGraph(rules_to_s_expr(fn))
    for it in range(max_iter):
        apply_rewrite_rules(eg, rules)
```

We employ the Graphviz API to visualize the e-classes, e-nodes, and their connections. Several visualization example is shown in section F.

A.4 Expression Extraction

Cost-based extraction retrieves a simplified expression by minimizing a user-defined cost over operators. The procedure returns an expression tree with the lowest total cost, where the cost is computed across all operators. Each e-class is assigned a cost equal to that of its cheapest e-node, and the cost of an e-node is defined recursively as

$$\text{cost(enode)} = \text{cost(operator)} + \sum_{\text{child e-classes}} \text{cost(child)}. \tag{5}$$

The algorithm iteratively updates costs for all e-nodes until convergence, i.e., when no further changes occur. This strategy yields short, simplified expressions and has been used in prior work (de Franca & Kronberger, 2025; de França & Kronberger, 2025), which employed e-graphs to simplify overly complex expressions in genetic programming. It is consistent with the philosophical principle of Occam's razor, which favors simpler expressions over more complex ones.

Random walk-based sampling, which draws a batch of expressions randomly from the e-graph. Starting from an e-class with no incoming edges, we randomly select an e-node within the current e-class and transition to the e-classes connected to that e-node. This process continues until an e-node with no outgoing edges is reached. The obtained sequence of visited e-nodes corresponds to drawing a valid expression from the constructed e-graph.

A.5 Implementation Remark

Our implementation builds upon an existing code snippet[1]. The most relevant prior implementation we identified is available in Haskell[2], the language in which e-graphs were originally developed. However, this choice complicates integration with Python-based environments, especially when interfacing with learning algorithms implemented in Python. A Python wrapper for Haskell egglog also exists[3], but its API is cumbersome and lacks clarity, limiting its usability for practical applications.

While packages such as `SymPy` provide APIs like `simplify` and `factor`, these functions map expressions to a fixed canonical form and do not support the richer class of rewrites and transformations. These APIs can return a smaller set of symbolically equivalent expressions than our EGG framework. Hence, they are not used in our implementation.

[1] https://colab.research.google.com/drive/1tNOQijJqe5tw-Pk9iqd6HHb2abC5aRid
[2] https://github.com/folivetti/eggp
[3] https://github.com/egraphs-good/egglog-python

B PROOF OF THEOROEM 3.1

B.1 PROBLEM SETTINGS

Necessary Definitions of Markov Decision Process. Consider a deterministic, finite-horizon Markov Decision Process (MDP) with state space $\mathcal{S}$ and action space $\mathcal{A}$. At each stage $t \in \{0, 1, \ldots, H-1\}$ the agent observes its current state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$, and deterministically transitions to the next state $s_{t+1} = f(s_t, a_t)$ while receiving a bounded reward $r_t \in [0, 1]$. Let H denote the maximum planning horizon. The total discounted reward over the H-step horizon is $\sum_{t=0}^{H-1} \gamma^t r_t$, where $\gamma \in (0, 1)$ is the discount factor. For any state–action pair (s, a), the *state–action value* is defined as $Q(s, a) \coloneqq \max_\pi \mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{H-1} \gamma^t r_t \middle| s_0 = s,\, a_0 = a\right]$, where π is any policy and $\tau = (s_0, a_0, \ldots, s_{H-1}, a_{H-1})$ is a trajectory generated by following π. Because the MDP is deterministic, the expectation is taken only over the agent's policy-induced action sequence. The *optimal value* of a state s is $V(s) \coloneqq \max_{a\in\mathcal{A}} Q(s, a)$.

Extensions to Symbolic Regression. In our symbolic regression setting, a state corresponds to a sequence of production rules that define a partially constructed expression, or equivalently, to the class of mathematical expressions that can be generated by completing this partially-complete expression. An action represents the selection of an available production rule that extends the current expression. The state space is the set of all possible expressions of maximum length H, and the action space is the set of production rules. We further provide concrete instantiations of this MDP for our Monte Carlo tree search implementation in Appendix D.1 and for deep reinforcement learning in Appendix D.2.

A *planning algorithm* is any procedure that, given a model of the MDP and a limited computational budget, uses simulated rollouts to estimate the value of available actions and to recommend a single action for execution. Suppose that after n simulations of a planning algorithm, the agent recommends an action a_T to execute at the current state s. Its estimated value is $Q(s, a_T)$, obtained by evaluating the discounted return that results from first taking a_T and then following an optimal policy for the remaining steps. The *simple regret* of this recommendation is

$$\texttt{regret}(T) \coloneqq V(s) - Q(s, a_T), \tag{6}$$

which quantifies the *loss in discounted return* incurred by choosing a_T instead of the truly optimal action. A smaller $\texttt{regret}(n)$ indicates that the planning algorithm used its limited simulation Budget more effectively to identify a near-optimal action.

Planning involves selecting promising transitions to simulate at each iteration, in order to recommend actions that minimize regret $\texttt{regret}(n)$. A popular strategy is the *optimism in the face of uncertainty* (OFU) principle, which explores actions by maximizing an upper confidence bound on the value function V.

In the context of symbolic regression, the planning task is therefore to sequentially apply production rules so as to construct a complete expression whose predicted outputs best match the target data, while respecting the grammar and structural constraints of the search space.

Definition 1 (Difficulty measure)**.** *The* near-optimal branching factor κ *of an MDP is defined as*

$$\kappa \coloneqq \limsup_{h\to\infty} |T_h^\infty|^{1/h} \in [1, K], \tag{7}$$

where $T_h^\infty \coloneqq \left\{a \in \mathcal{A}^h : V^* - V(a) \le \frac{\gamma^h}{1-\gamma}\right\}$ *is the set of near-optimal nodes at depth* h*.*

In standard MCTS, the root of the search tree corresponds to the initial state $s_0 \in S$. A node at depth h represents an action sequence $a = (a_1, \ldots, a_h) \in \mathcal{A}^h$, which leads to a terminal state $s(a)$. The search tree at iteration n is denoted $\texttt{tree}(T)$, and its maximum expanded depth is d_n.

Historical research has shown that the UCT principle suffers from a theoretical worst case in which it is difficult to derive meaningful regret bounds, making it unsuitable for analyzing the benefit of EGG over standard MCTS. Therefore, we assume that MCTS operates under the Optimistic Planning for Deterministic Systems (OPD) framework Leurent & Maillard (2020) rather than the UCT principle.

Theorem B.1 (Regret bound of MCTS (Munos (2014), chapter 5)). *Let* $\texttt{regret}(n)$ *denote the simple regret after* n *iterations of OPD used in MCTS. Then, we have:*

$$\texttt{regret}(T) = \widetilde{\mathcal{O}}\Big(n^{-\frac{\log(1/\gamma)}{\log \kappa}}\Big). \tag{8}$$

Here, the soft-O version of big-O notation is defined as: $f_n = \widetilde{\mathcal{O}}(n^{-\alpha})$ *means that decays at least as fast as* $n^{-\alpha}$*, up to a polylog factor.*

When κ is small, only a limited number of nodes must be explored at each depth, allowing the algorithm to plan deeper, given a budget of n simulations. The quantity κ represents the branching factor of the subtree of near-optimal nodes that can be sampled by the OPD algorithm, serving as an *effective* branching factor in contrast to the true branching factor K. Hence, κ directly governs the achievable simple regret of OPD (and its variants) on a given MDP.

The theoretical analysis of EGG-MCTS is built on top of the analysis procedure in Leurent & Maillard (2020). Unlike their analysis, which requires unrolling the graph into a tree and addressing potentially infinite-length paths induced by cycles, our analysis starts from the unrolled tree derived from the graph in Leurent & Maillard (2020).

Definition 2 (Upper and Lower bounds of the Value function). *Let* $\texttt{tree}(T)$ *be the search tree after* T *expansions. Each node corresponds to an action sequence* $(a_0, a_1, \ldots, a_{H-1})$ *from the root, and let* s *denote the MDP state reached by executing this sequence from the root state* s_0*. The true value associated with this node is the optimal value of the reached state, denoted by* $V(s)$*. A pair of functions* (L_T, U_T) *is said to provide* lower and upper bounds *for* V *on* $\texttt{tree}(T)$ *if*

$$L_T \le V(s) \le U_T, \qquad \textit{for internal node } s \textit{ in the tree } \texttt{tree}(T)$$

Definition 3 (Finer Difficulty Measure). *We define the* near-optimal branching factor *associated with bounds* (L_T, U_T) *as*

$$\kappa(L_T, U_T) \coloneqq \limsup_{h\to\infty} \left|T_h^{\infty}(L_T, U_T)\right|^{1/h} \in [1, K], \tag{9}$$

where $T_h^{\infty}(L, U) \coloneqq \Big\{a \in \mathcal{A}^h : V^* - V(a) \le \gamma^h\big(U(a) - L(a)\big)\Big\}$.

Since (L_T, U_T) tighten with T, the sequence of bounds $\big(L_T, U_T\big)_{T\ge 0}$ is non-increasing, i.e.,

$$0 \le \cdots \le L_{T-1} \le L_T \le V \le U_T \le U_{T-1} \le \cdots \le V_{\max}$$

they will finally converges to a limit $\kappa_\infty = \lim_{n\to\infty} \kappa(L_T, U_T)$.

Theorem B.2 (Regret Bound of EGG-MCTS). *Let* $\kappa_T = \kappa(L_T, U_T)$*, and define* $\kappa_\infty = \lim_{n\to\infty} \kappa(L_T, U_T) \in [1, K]$*. Then* EGG-*MCTS achieves the regret bound*

$$\texttt{regret}(T) = \widetilde{\mathcal{O}}\Big(n^{-\frac{\log(1/\gamma)}{\log \kappa_\infty}}\Big), \qquad \textit{where } \kappa_\infty \le \kappa \tag{10}$$

Proof. The analysis is similar to that of Leurent & Maillard (2020, Theorem 16). In their approach, the graph is *unrolled* into a tree to leverage the analysis of Theorem B.1. This unrolled tree contains every action sequence that can be traversed in G_n. There would exist a path of infinite length if the graph contains cycles. The behavior of the graph G_n is therefore analyzed through its corresponding unrolled tree $\texttt{UnrollTree}(T)$.

Our EGG-MCTS behaves almost identically to this unrolled tree, except that it contains no paths of infinite length, since our algorithm cannot empirically generate or update along an infinite action sequence. Transporting the graph-based value bounds (L_T, U_T) to the unrolled tree and applying the depth–regret argument of Leurent & Maillard (2020, Appendix A.9) yields the rate in equation 10. The detailed derivation is therefore omitted here for brevity. □

Remark 1. *For a class of symbolic regression problems where a large set of mathematical identities is applicable, the resulting overlaps among action paths can be extensive. In this case, the effective branching factor* κ_∞ *can be much smaller than the nominal branching factor* κ*, leading to substantially tighter regret guarantees.*

C Proof of Theorem 3.2

C.1 Notation and Assumptions

Let τ be a sequence of production rules sampled from a sequential decoder with probability $p_\theta(\tau)$. ϕ is the expression constructed by τ following grammar definition. Let $\mathcal{S}_\phi \subseteq \Pi$ be the equivalence class (under the rewrite system $\mathcal{R}$) of sequences that deterministically construct ϕ or can be rewritten into ϕ by applying rewrite rules in $\mathcal{R}$. For notation simplicity and clarity, we define the probability over the set of sequences $\mathcal{S}_\phi$

$$q_\theta(\phi) \coloneqq \sum_{\tau \in \mathcal{S}_\phi} p_\theta(\tau). \tag{11}$$

The reward of the expression ϕ is defined as the goodness-of-fit on the dataset D.

C.2 Proof of Unbiasedness

Lemma 1 (Key identity)**.** *For any $\phi \in \Phi$ with $q_\theta(\phi) > 0$, $\mathbb{E}_{\tau \sim p_\theta}\left[\nabla_\theta \log p_\theta(\tau) | \phi\right] = \nabla_\theta \log q_\theta(\phi)$.*

Proof. By the definition of conditional probability, $\mathbb{P}(\tau|\phi) = \frac{p_\theta(\tau)}{q_\theta(\phi)}$, for $\tau \in \mathcal{S}_\phi$ and 0 otherwise. Hence,

$$\begin{aligned}
\mathbb{E}\left[\nabla_\theta \log p_\theta(\tau) \,|\, \phi\right] = \sum_{\tau \in \mathcal{S}_\phi} \nabla_\theta \log p_\theta(\tau)\, \frac{p_\theta(\tau)}{q_\theta(\phi)} = \frac{1}{q_\theta(\phi)} \sum_{\tau \in \mathcal{S}_\phi} \nabla_\theta p_\theta(\tau) = \frac{1}{q_\theta(\phi)} \nabla_\theta \sum_{\tau \in \mathcal{S}_\phi} p_\theta(\tau) \\
= \frac{\nabla_\theta q_\theta(\phi)}{q_\theta(\phi)} \\
= \nabla_\theta \log q_\theta(\phi).
\end{aligned}$$

□

Proposition 1 (Unbiasedness)**.** $\mathbb{E}_{\tau \sim p_\theta}\left[g(\theta)\right] = \mathbb{E}_{\tau \sim p_\theta}\left[g_{\texttt{egg}}(\theta)\right]$.

Proof. First, for the EGG-based estimator,

$$\begin{aligned}
\mathbb{E}\left[g_{\texttt{egg}}(\theta)\right] = \sum_{\tau \in \Pi} \texttt{reward}(\phi)\, \nabla_\theta \log q_\theta(\phi)\; p_\theta(\tau) = \sum_{\phi \in \Phi} \sum_{\tau \in \mathcal{S}_\phi} \texttt{reward}(\phi)\, \nabla_\theta \log q_\theta(\phi)\; p_\theta(\tau) \\
= \sum_{\phi \in \Phi} \texttt{reward}(\phi)\, \nabla_\theta \log q_\theta(\phi) \underbrace{\sum_{\tau \in \mathcal{S}_\phi} p_\theta(\tau)}_{= q_\theta(\phi)} \\
= \sum_{\phi \in \Phi} \texttt{reward}(\phi)\, \nabla_\theta q_\theta(\phi) \\
= \nabla_\theta \sum_{\phi \in \Phi} \texttt{reward}(\phi)\, q_\theta(\phi).
\end{aligned}$$

For the standard estimator,

$$\begin{aligned}
\mathbb{E}\left[g(\theta)\right] = \sum_{\tau \in \Pi} \texttt{reward}(\tau)\, \nabla_\theta \log p_\theta(\tau)\; p_\theta(\tau) = \sum_{\tau \in \Pi} \texttt{reward}(\tau)\, \nabla_\theta p_\theta(\tau) \\
= \nabla_\theta \sum_{\tau \in \Pi} \texttt{reward}(\tau)\, p_\theta(\tau) \\
= \nabla_\theta \sum_{\phi \in \Phi} \sum_{\tau \in \mathcal{S}_\phi} \texttt{reward}(\phi)\, p_\theta(\tau) \\
= \nabla_\theta \sum_{\phi \in \Phi} \texttt{reward}(\phi)\, q_\theta(\phi),
\end{aligned}$$

where we used class-invariance of the reward and equation 11. This equals the expression obtained for $g_{\texttt{egg}}$, proving unbiasedness. □

C.3 Proof of Variance Reduction

Define the σ-field generated by the expression ϕ as $\sigma(\phi)$. Consider the random variable

$$Z \coloneqq \texttt{reward}(\tau)\,\nabla_\theta \log p_\theta(\tau) = \texttt{reward}(\phi)\,\nabla_\theta \log p_\theta(\tau),$$

where the equality uses reward invariance within each $\mathcal{S}_\phi$. By Lemma 1,

$$\mathbb{E}\ [\,Z \mid \phi] = \texttt{reward}(\phi)\,\mathbb{E}\ [\,\nabla_\theta \log p_\theta(\tau) \mid \phi] = \texttt{reward}(\phi)\,\nabla_\theta \log q_\theta(\phi) = g_{\texttt{egg}}(\theta).$$

Thus, $g_{\texttt{egg}}(\theta)$ is the *Rao–Blackwellization* of $g(\theta)$ with respect to $\sigma(\phi)$ (Casella & Robert, 1996).

Proposition 2 (Variance reduction)**.** $\mathrm{Var}\left(g_{\texttt{egg}}(\theta)\right) \ \le\ \mathrm{Var}\left(g(\theta)\right)$.

Proof. By the law of total variance,

$$\mathrm{Var}(Z) = \mathrm{Var}\big(\mathbb{E}[Z \mid \phi]\big) \ +\ \mathbb{E}\big[\mathrm{Var}(Z \mid \phi)\big] \ \ge\ \mathrm{Var}\ \big(\mathbb{E}[Z \mid \phi]\big).$$

Substituting $Z = g(\theta)$ and $\mathbb{E}[Z \mid \phi] = g_{\texttt{egg}}(\theta)$ yields

$$\mathrm{Var}\left(g(\theta)\right) \ge \mathrm{Var}\left(g_{\texttt{egg}}(\theta)\right).$$

□

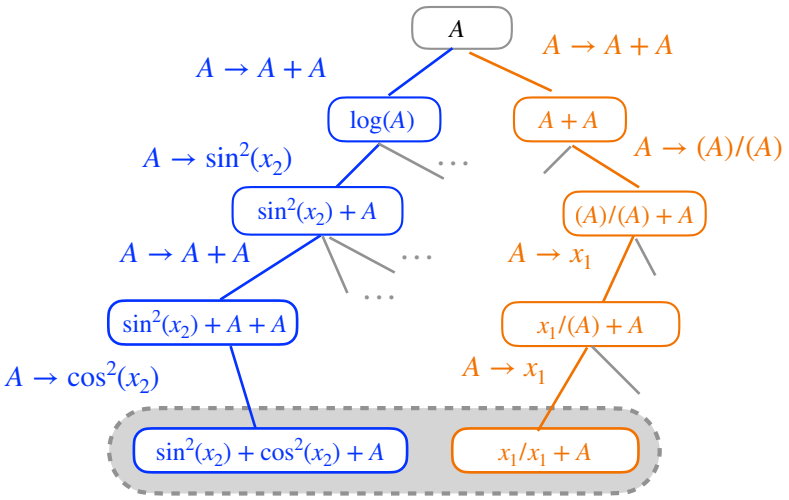
Combining Propositions 1 and 2 proves Theorem 3.2.

Remark on baselines. If a baseline $b(\cdot)$ independent of the sampled action (sequence) is included, both estimators remain unbiased, and the Rao–Blackwell argument applies to the centered variable as well, so the variance reduction still holds (and can be further improved by an appropriate baseline choice).

D Implementation of Egg-embedded Symbolic Regression

D.1 Implementation of Egg-MCTS

Markov Decision Process Definition for MCTS. We model the search process as a finite-horizon Markov decision process (MDP). A state corresponds to a partially constructed expression, and each node in the MCTS search tree represents one such state. An action corresponds to the application of a single production rule to the first non-terminal symbol in the current state. The transition dynamics are deterministic: given a state–action pair, the successor state is uniquely determined by the application of the selected production rule. Rewards are assigned only at terminal states (leaf nodes of the search tree), obtained by evaluating the completed expression on the dataset according to an evaluation metric. The episode length is thus equal to the number of applied production rules, and we set the discount factor to 1. A trajectory is a sequence of state–action transitions corresponding to a path from the root node to a leaf node in the search tree, resulting in either a valid or an invalid expression. The MDP horizon is finite: each episode terminates when the path from the root corresponds to a valid expression with no non-terminal symbols or when the maximum depth H is reached.



(a) It corresponds to an underlying function $\phi = 1 + A$, with the rewrite rule "$\sin^2(a) + \cos^2(a) \rightsquigarrow 1$"

(b) It corresponds to a underlying function $\phi = 1 + A$, with the rewrite rule:"$a/a \rightsquigarrow 1$".

Figure 8: The two paths to the leaf node correspond to the same underlying function $\phi = 1 + A$, where A is a placeholder for an arbitrary sub-expression. This equivalence is derived using the rewrite rules $\sin(a) + \cos(a) \rightsquigarrow 1$ and $a/a \rightsquigarrow 1$.

Example 2. *An example execution pipeline of* Egg-*MCTS is provided in Figure 8. In Figure 8, we examine two distinct paths within the search tree maintained by the algorithm:*

$$\textit{Path } 1 : (A \to A + A, A \to \sin^2(x_2), A \to A + A, A \to \cos^2(x_2)),$$
$$\textit{Node } 1 : \; s_1 = \sin^2(x_2) + \cos^2(x_2) + A.$$
$$\textit{Path } 2 : (A \to A + A, A \to (A)/(A), A \to x_1, A \to x_1),$$
$$\textit{Node } 2 : \; s_2 = x_1/x_1 + A.$$

Here, each path is a sequence of edge labels from the root to the leaf node s_i. Based on the grammar definition in section 2, node s_1 corresponds to the partially completed expression $\sin^2(x_2) + \cos^2(x_2) + A$, while node s_2 represents $x_1/x_1 + A$. The two nodes are equivalent under the rewrite rule $\sin(a) + \cos(a) \rightsquigarrow 1$ and $a/a \rightsquigarrow 1$. Similar to Example 1, their rewards, estimating the averaged goodness-of-fit of expressions on the dataset, should be approximately equal:

$$\texttt{Reward}(s_1, a) \approx \texttt{Reward}(s_2, a), \qquad \forall \textit{ rule } a \in \textit{ the set of available rules}$$

Exploring sub-trees of s_1 and s_2 independently leads to redundant computation, as MCTS repeatedly traverses equivalent regions of the search space.

The algorithmic procedure of EGG-MCTS is described in Section 3.2. The corresponding visualized pipeline is provided in Figure 3. The code implementation is adapted from existing MCTS frameworks [4].

Connection to Existing Research. In many applications, multiple action sequences may lead to the same state, resulting in duplicate nodes within the search tree. To mitigate this redundancy, Saffidine et al. (2012) proposed the use of a transposition table in the context of Go, enabling the reuse of information from previous updates. Similarly, Leurent & Maillard (2020) introduced *Monte Carlo Graph Search* (MCGS), which replaces the search tree with a graph that merges identical states. In MCGS, nodes correspond to unique states $s \in \mathcal{S}$ and edges represent state transitions, with the root node corresponding to the initial state s_0. At iteration n, if executing action a from state s leads to a successor state s' that already exists in the graph, no new node is created.

Our EGG-MCTS adopts the same principle of detecting identical states during search but avoids explicit graph construction. The constructed e-graph serves as a transposition table that stores and detects identical states. Following the same update rule used in a transposition table, EGG-MCTS simultaneously updates all paths in the tree that lead to the selected state. This design preserves compatibility with standard MCTS implementations while achieving a theoretical acceleration comparable to that of Leurent & Maillard (2020).

D.2 IMPLEMENTATION OF EGG-DRL

Markov Decision Process Formulation for DRL. We model this problem as an RL agent that search for the optimal sequence of grammar rules. At decoding step t, the agent samples a rule τ_t conditioned on the previously selected rules $\tau_1, \ldots, \tau_{t-1}$. The sequential decoder, parameterized by θ, defines a stochastic policy $\pi_\theta(\tau_t \mid \tau_1, \ldots, \tau_{t-1}) := p_\theta(\tau_t \mid \tau_1, \ldots, \tau_{t-1})$. We define the *state* at step t as the prefix of sampled rules $s_t := (\tau_1, \ldots, \tau_{t-1})$, and the grammar rules in the output vocabulary constitute the *action space* for the RL agent. The environment transition is deterministic: applying action τ_t in state s_t yields the next state $s_{t+1} = (\tau_1, \ldots, \tau_t)$.

Rewards are obtained by evaluating the completed expression on the dataset according to a chosen evaluation metric (e.g., $1/(1 + \texttt{NMSE}(\phi))$). The MDP has a finite horizon: an episode terminates once the output sequence corresponds to a valid expression with no non-terminal symbols, or when the maximum sequence length H is reached. The objective of the RL agent is to learn a policy that selects sequences of grammar rules so as to maximize the expected reward.

Configuration of Sequential Decoder. The sequential decoder can be implemented using various architectures, such as RNNs (Salehinejad et al., 2017), GRUs (Chung et al., 2014), LSTMs (Greff et al., 2016), or decoder-only Transformers (Vaswani et al., 2017). The input and output vocabularies for the decoder consist of grammar rules that encode variables, coefficients, and mathematical operators. Figure 9(a) illustrates an example of the output vocabulary.

At each time step, the model predicts a categorical distribution over the next rule, conditioned on the previously generated rules. At step t, the decoder (denoted as "`SequentialDecoder`") takes the previously generated rules $(\tau_1, \ldots, \tau_{t-1})$ and the hidden state $\mathbf{h}_{t-1}$, and computes

$$\begin{aligned} \mathbf{h}_t &= \texttt{SequentialDecoder}(\tau_{t-1}, \mathbf{h}_{t-1}), \\ \mathbf{z}_t &= \mathbf{h}_t W_o + b_o, \\ p_\theta(\tau_t \mid \tau_1, \ldots, \tau_{t-1}) &= \texttt{softmax}(\mathbf{z}_t), \end{aligned}$$

where $W_o \in \mathbb{R}^{d \times |V|}$ is the output weight matrix, $b_o \in \mathbb{R}^{|V|}$ is the bias vector, and $|V|$ is the size of the vocabulary. The next rule τ_t is then sampled from this distribution, $\tau_t \sim p_\theta(\tau_t \mid \tau_1, \ldots, \tau_{t-1})$, and fed back into the decoder as input for the next step, until the sequence is completed. After H steps, the full sequence $\tau = (\tau_1, \ldots, \tau_H)$ is generated with probability $p_\theta(\tau) = \prod_{t=1}^{H} p_\theta(\tau_t \mid \tau_1, \ldots, \tau_{t-1})$, with the convention that τ_1 is a special start symbol.

Each sequence τ is then converted into an expression following the grammar definition in Section 2. If the sequence terminates prematurely, grammar rules for variables or constants are appended at random to complete the expression. Conversely, if a valid expression is obtained before the sequence

[4] `https://github.com/isds-neu/SymbolicPhysicsLearner`

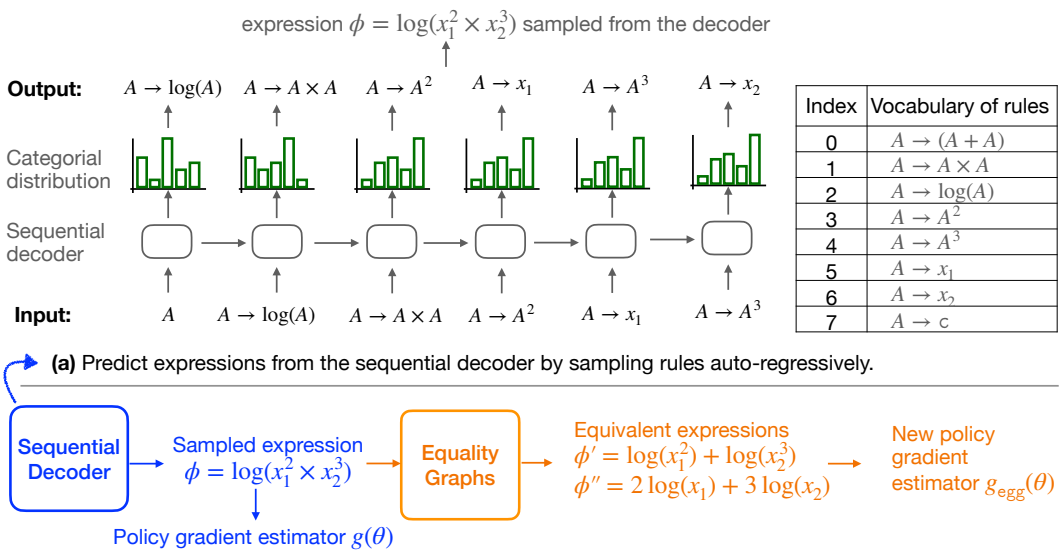


Figure 9: Framework of DRL and our EGG-DRL. **(a)** The sequential decoder autoregressively samples grammar rules according to the modeled probability distribution. **(b)** In classic DRL, the sampled expressions are directly used to compute the policy gradient estimator $g(\theta)$. While EGG-DRL constructs e-graphs from the sampled expressions, extracts symbolic variants, and computes the revised estimator $g_{\text{egg}}(\theta)$.

ends, the remaining rules are discarded. In both cases, the probability $p_\theta(\tau)$ is updated consistently with the applied modifications.

Finally, the model parameters are updated using gradient-based optimization (e.g., the Adam optimizer) with either the classic policy gradient estimator $g(\theta)$ or the proposed EGG-based estimator $g_{\text{egg}}(\theta)$. The whole pipeline is visualized in Figure 9.

Connection to Existing Research. Several techniques have been introduced to reduce the variance of policy gradient estimates. One widely used approach is the control variate method, where a baseline is subtracted from the reward to stabilize the gradient (Weaver & Tao, 2001). Another idea is Rao-Blackwellization (Casella & Robert, 1996). Other approaches, such as reward reshaping (Zheng et al., 2018), modify rewards for specific state-action pairs. Inspired by stochastic variance-reduced gradient methods (Johnson & Zhang, 2013; Deng et al., 2021), Papini et al. (2018) proposed a variance-reduction technique tailored for policy gradients.

Unlike these existing methods, our proposed EGG is the first to reduce variance through the domain knowledge of the symbolic regression application, and show a seamless integration into the learning framework and attain reduced variance.

D.3 IMPLEMENTATION OF EGG-LLM

Our EGG is adopted on top of the original LLM-SR (Shojaee et al., 2025).

(a) Hypothesis Generation. The LLM generates multiple candidate expressions based on a prompt describing the problem background and the definitions of each variable.

(b) Data-driven Evaluation. Each candidate expression is evaluated based on its fitness on the training dataset.

(c) **Inferring Equivalent Expressions.** For ease of integration with the e-graph system, we convert the generated Python functions into lambda expressions. This simplifies their manipulation and evaluation during equivalence analysis.

```
# Original function format
```

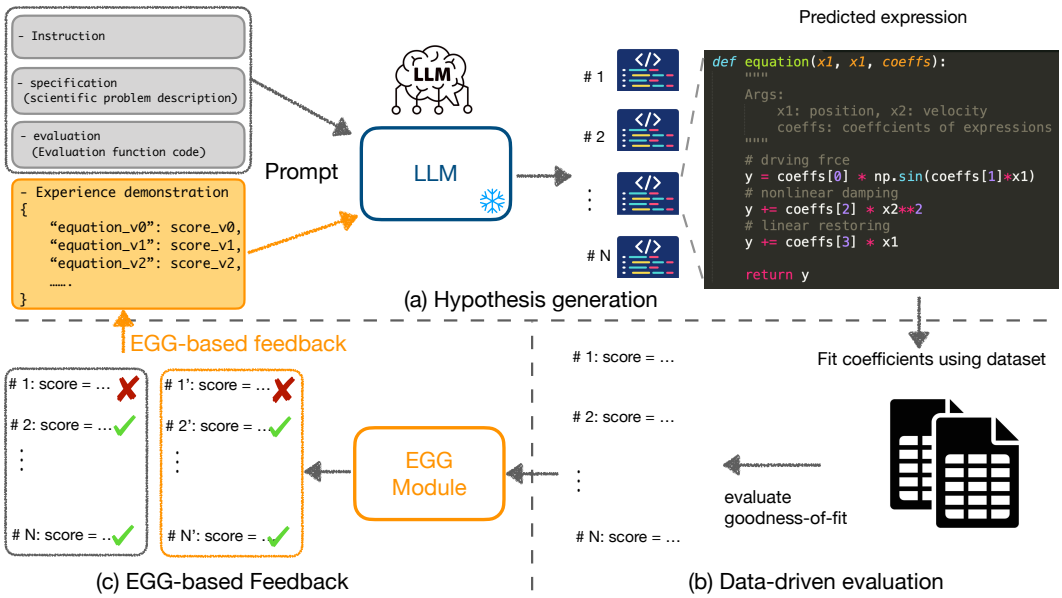


Figure 10: Pipeline of EGG-LLM. **(a) Hypothesis Generation.** The LLM receives prompts derived from the problem specification and predicts multiple candidate expressions. **(b) Data-Driven Evaluation.** The coefficients of each candidate are fitted to the dataset and then evaluated on a separate validation set to compute a goodness-of-fit score. **(c) EGG-Based Feedback.** The proposed EGG module generates symbolically equivalent variants, which are fed back into step (a) to guide the next round of hypothesis generation.

```
def hypothesis(x1, x2, coeffs):
    expr = x1 * coeffs[0] + x2 * coeffs[0]
    return expr

# Converted into lambda format
hypothesis = lambda x1, x2, coeffs: x1 * coeffs[0] + x2 * coeffs[0]
```

This conversion allows us to transform the hypothesized Python function into an expression tree directly. For each fitted expression, we construct an e-graph and apply a predefined set of rewrite rules until reaching a fixed iteration limit. From the resulting e-graph, we sample K unique equivalent expressions. Each sampled expression is then converted back into a Python function to facilitate further interaction with the LLM.

```
# obtained equivalent expression
equiva_seq = ["A->log(A)", "A->A*A", "A->x1", "A->x2"]
# Converted function format
def equiva_hypothesis(x1, x2, coeffs):
        return x1 * x2
```

(d) **EGG-based Feedback.** In subsequent iterations, the LLM receives feedback consisting of previously generated expressions along with their fitness scores. Our feedback is enriched with both the original hypotheses and their equivalent expressions derived via EGG module, enabling the model to refine future generations. High-performing expressions are retained and further updated over multiple rounds.

D.4 Final Remark

The E2E-Transformer framework (Kamienny et al., 2022) is structurally very similar to the neural network used in the DRL model, with the main difference being that it is trained using a cross-entropy loss function. A straightforward way to integrate the EGG module into E2E-Transformer is through data augmentation, where additional training examples are generated using EGG.

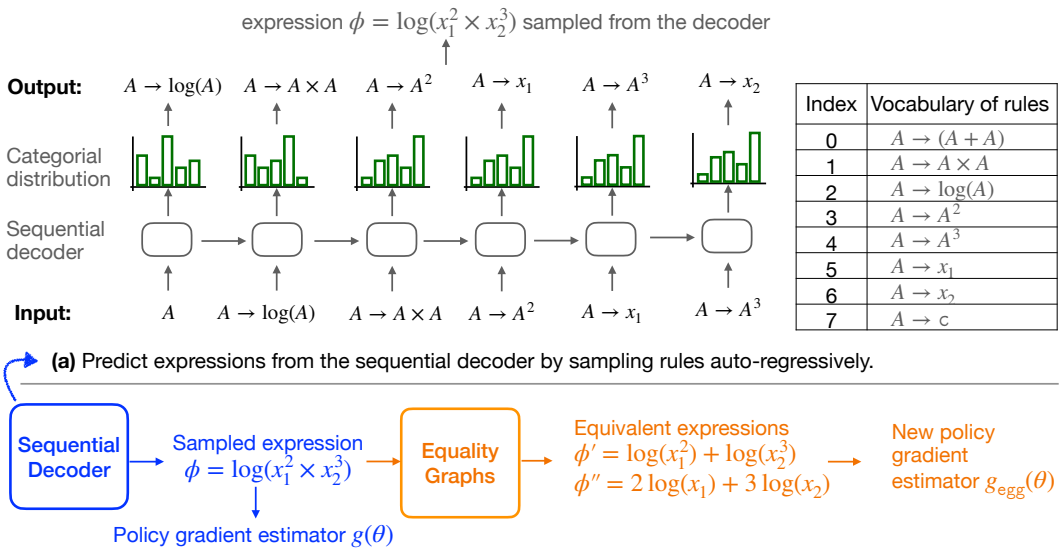


Figure 11: Framework of DRL and our EGG-SymNet. **(a)** The sequential decoder autoregressively samples grammar rules according to the modeled probability distribution. **(b)** In classic DRL, the sampled expressions are directly used to compute the policy gradient estimator $g(\theta)$. While EGG-DRL constructs e-graphs from the sampled expressions, extracts symbolic variants, and computes the revised estimator $g_{\text{egg}}(\theta)$.

To date, only one work has explored the use of a text-based diffusion model for symbolic regression (Bastiani et al., 2025). However, its implementation has not been released publicly. Moreover, the proposed pipeline relies on multiple interconnected components, which makes it difficult to reproduce or isolate for independent evaluation. For these reasons, we do not include a comparison between the text-based diffusion approach and EGG in this paper.

Several extensions of the baseline considered in this study offer empirical improvements but lack theoretical guarantees. For example, Mundhenk et al. (2021a) employ genetic programming to refine the predictions of the DRL model, and Tenachi et al. (2023) introduce unit constraints in physical systems to eliminate infeasible expressions. Integrating EGG with these variants is an interesting direction that we leave for future investigation.

E EXPERIMENT SETTINGS

E.1 BASELINES AND HYPER-PARAMETERS CONFIGURATIONS

The representative baselines are selected as they all regard the search for the optimal expression as a sequential decision-making process. A detailed description of each method is provided below:

MCTS. Sun et al. (2023) propose to use the Monte Carlo tree search algorithm to explore the space of symbolic expressions defined by a context-free grammar (described in Section A) to find high-performing expressions. Despite being implemented similarly, this method appears under various names in previous works (Todorovski & Dzeroski, 1997; Ganzert et al., 2010; Brence et al., 2021; Sun et al., 2023; Kamienny et al., 2023). We choose the implementation [5] from Sun et al. (2023) and serve as a representative of this family of methods.

DRL. Petersen et al. (2021) propose to use a recurrent neural network (RNN) trained via a (risk-seeking) reinforcement learning objective. The RNN sequentially generates candidate expressions and is optimized using a risk-seeking policy gradient to encourage the discovery of high-quality expressions. We chose this implementation [6]. We use three types of sequential decoders for the time benchmark setting. The major configurations are listed in Table 4.

Table 4: Hyperparameters for the DRL Model with different types of neural network. This configuration is also used in Figure 6.

General Parameters		
max length of output sequence	20	
batch size of generated sequence	1024	
total learning iterations	200	
Reward function	$\frac{1}{1+\texttt{NMSE}(\phi)}$	
Optimizer Hyperparameters		
optimizer	Adam	
learning rate	0.009	
entropy weight	0.03	
entropy gamma	0.7	
Decoder-relevant Hyperparameters		
choice of decoder	LSTM	Decoder-only Transformer
num of layers	3	3
hidden size	128	128
dropout rate	0.5	/
number of head	/	6

LLM-SR. Shojaee et al. (2025) proposed to use pretrained large language models, such as GPT3.5, to generate symbolic expressions based on task-specific prompts. The expressions are evaluated and iteratively refined over multiple rounds. We choose their implementation at [7].

For each baseline, we adopted the most straightforward implementation. Only minimal modifications were made to ensure that all baselines use the same input data and problem configurations, and are compatible with the proposed EGG module. We also rename the abbreviation of each method, to clearly present the core idea in each method and uniformly present the adaptation of our EGG on top of these baselines.

Expression-related Configurations. When fitting the values of open constants in each expression, we sample a batch of data with batch size 1024 from the data Oracle. The open constants in the expressions are fitted on the data using the BFGS optimizer[8]. We use a multi-processor library to fit multiple expressions using 8 CPU cores in parallel. This greatly reduced the total training time.

[5] https://github.com/isds-neu/SymbolicPhysicsLearner

[6] https://github.com/dso-org/deep-symbolic-optimization

[7] https://github.com/deep-symbolic-mathematics/LLM-SR

[8] https://docs.scipy.org/doc/scipy/reference/optimize.minimize-bfgs.html

Table 5: Hyper-parameter configurations for symbolic expression computation.

Expressions and Dataset	
training dataset size	2048
validation and testing dataset size	2048
coefficient fitting optimizer	BFGS
maximum allowed coefficients	20
optimization termination criterion	error is less than $1e-6$

An expression containing a placeholder symbol A or containing more than 20 open constants is not evaluated on the data; its fitness score is $-\infty$.

The normalized mean-squared error metric is further defined in Equation 12.

To ensure fairness, we use an interface, called `dataOracle`, which returns a batch of (noisy) observations of the ground-truth equations. To fast evaluate the obtained expression is evaluated using the Sympy library, and the step for fitting open constants in the expression with the dataset uses the optimizer provided in the Scipy library[9].

During testing, we ensure that all predicted expressions are evaluated on the same testing dataset by configuring the `dataOracle` with a fixed random seed, which guarantees that the same dataset is returned for evaluation.

We further summarize all the above necessary configurations in Table 5.

E.2 Evaluation Metrics

We mainly consider two evaluation criteria for the learning algorithms tested in our work: 1) the goodness-of-fit measure and 2) the total running time of the learning algorithms.

The goodness-of-fit indicates how well the learning algorithms perform in discovering unknown symbolic expressions. Given a testing dataset $D_{\text{test}} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ generated from the ground-truth expression ϕ, we measure the goodness-of-fit of a predicted expression $\phi_{\texttt{pred}}$, by evaluating the mean-squared-error (MSE) and normalized-mean-squared-error (NMSE):

$$\begin{aligned} \text{MSE} &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \phi_{\texttt{pred}}(\mathbf{x}_i))^2, \qquad \text{NMSE} = \frac{\frac{1}{n}\sum_{i=1}^{n}(y_i - \phi_{\texttt{pred}}(\mathbf{x}_i))^2}{\sigma_y^2} \\ \text{RMSE} &= \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \phi_{\texttt{pred}}(\mathbf{x}_i))^2}, \qquad \text{NRMSE} = \frac{1}{\sigma_y}\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \phi_{\texttt{pred}}(\mathbf{x}_i))^2} \end{aligned} \tag{12}$$

where the empirical variance $\sigma_y = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \frac{1}{n}\sum_{i=1}^{n} y_i\right)^2}$. Note that the coefficient of determination (R^2) metric (Nagelkerke et al., 1991; Cava et al., 2021) is equal to $(1 - \text{NMSE})$ and therefore omitted in the experiments.

[9] `https://docs.scipy.org/doc/scipy/tutorial/optimize.html`

F Extra Experimental Results

F.1 Visualization of Egg Construction

We summarize the list of example expressions, their symbolic-equivalent variants, and the visualized EGG in Table 6.

Table 6: Summary of expressions, their symbolic-equivalent variants, and the visualized o it visualization of more e-graphs obtained via EGG.

ID	Original equation	Symbolic-equivalent variant	E-graph visualization
1	$\log(x_1^3 x_2^2)$	$3\log(x_1) + 2\log(x_2)$	Figure 12
2	$\exp(c_1 x_1 + x_2)$	$\exp(c_1 x_1)\exp(x_2)$	Figure 13
3	$\sin(x_1 + x_2)$	$\sin(x_1)\cos(x_2) + \cos(x_1)\sin(x_1)$	Figure 14
4	$\log(x_1 \times \ldots x_n)$	$\log(x_1) + \ldots \log(x_n)$	Figure 16
5	$\sin(x_1 + \ldots + x_n)$	omitted	Figure 17

(a) Example symbolic regressions

ID	Original equation	Symbolic-equivalent variant	E-graph visualization
I.15.3x	$\frac{x_0 - x_1 x_2}{\sqrt{c^2 - x_1^2}}$	$\frac{x_0 - x_1 x_2}{\sqrt{c+x_1}\sqrt{c-x_1}}$	Figure 18
I.30.3	$x_0 \frac{\sin^2(x_1 x_2/2)}{\sin^2(x_2/2)}$	$x_0 \left(\frac{\sin(x_1 x_2/2)}{\sin(x_2/2)}\right)^2$	Figure 19
I.44.4	$c_1 x_0 x_1 \log(x_2/x_3)$	$c_1 x_0 x_1 (\log(x_2) - \log(x_3))$	Figure 20
I.50.26	$x_0(\cos(x_1 x_2) + x_3 \cos^2(x_1 x_2))$	$x_0 \cos(x_1 x_2)\,(1 + x_3 \cos(x_1 x_2))$	Figure 21
II.6.15b	$\frac{3x_0}{4\pi} \frac{3\cos x_1 \sin x_1}{x_2^3}$	$\frac{3x_0}{8\pi} \frac{\sin(2x_1)}{x_2^3}$	Figure 22
II.35.18	$\frac{x_0}{\exp(x_1 x_2/x_3) + \exp(-x_1 x_2/x_3)}$	$\frac{x_0^2}{2\cosh(x_1 x_2/x_3)}$	Figure 23
II.35.21	$x_0 x_1 \tanh(x_1 x_2/x_3)$	$x_0 x_1 \frac{\exp(2x_1 x_2/x_3) - 1}{\exp(2x_1 x_2/x_3) + 1}$	Figure 24

(b) selected complex symbolic regressions from Feynman Dataset.

E-graph for log **operator.** Figure 12 illustrates the e-graph construction for the expression $\log(x_1^3 x_2^2)$ using rewrite rules for log operator: $\log(a \times b) \rightsquigarrow \log(a) + \exp(b)$ and also $\log(a^b) \rightsquigarrow b \log a$. This visualization is based on the Graphviz API. We also label each e-class with a unique index for clarity. As shown in Figure 12, the saturated e-graph grows significantly larger as more rewrite rules are applied. In our full implementation, we incorporate many additional rewrite rules beyond these basic logarithmic identities, resulting in substantially larger and more complex e-graphs.

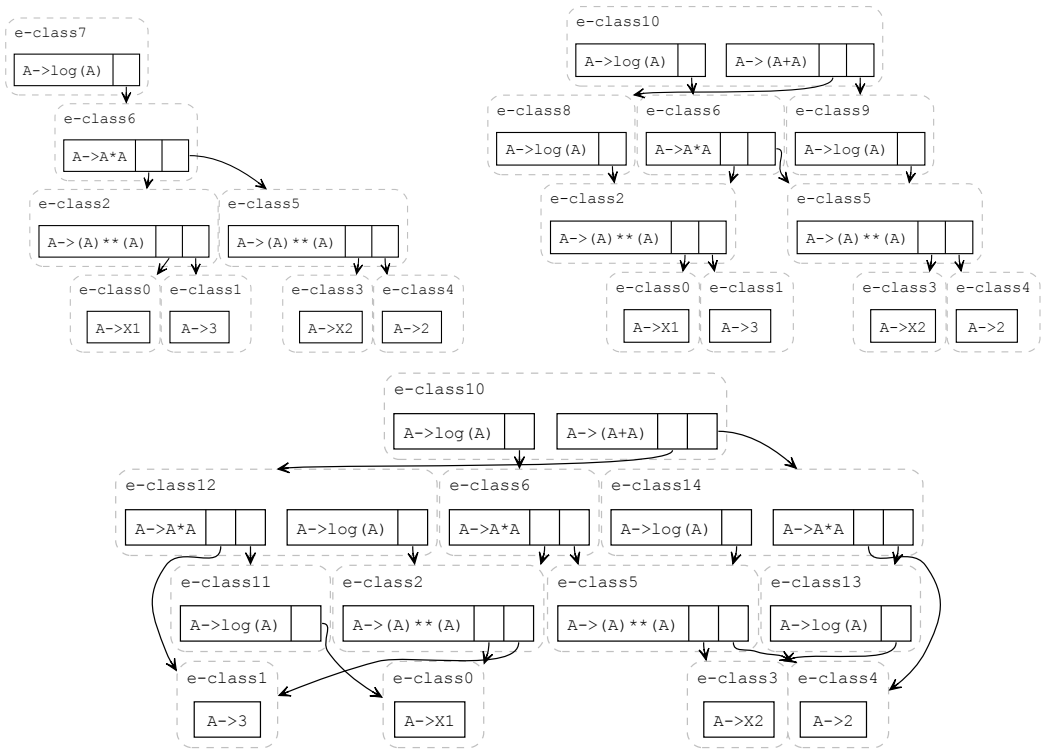


Figure 12: **(Top left)** Initialized e-graphs for expression $\log(x_1^3 x_2^2)$. **(Top Right)** The saturated e-graphs after applying one rewrite rule $\log(ab) \rightsquigarrow \log a + \log b$. **(Bottom)** The saturated e-graphs after applying two rewrite rules: $\log(ab) \rightsquigarrow \log a + \log b$ and $\log(a^b) \rightsquigarrow b \log a$.

E-graph for exp **operator.** Figure 13 illustrates the e-graph construction for the expression $\exp(c_1 x_1 + x_2)$ using the exponential rewrite rule $\exp(a + b) \rightsquigarrow \exp(a) \times \exp(b)$. The input expression $\exp(c_1 \times x_1 + x_2)$ is represented as a sequence of production rules: $(A \to \exp(A),\ A \to A + A,\ A \to A \times A,\ A \to c_1,\ A \to x_1,\ A \to x_2)$.

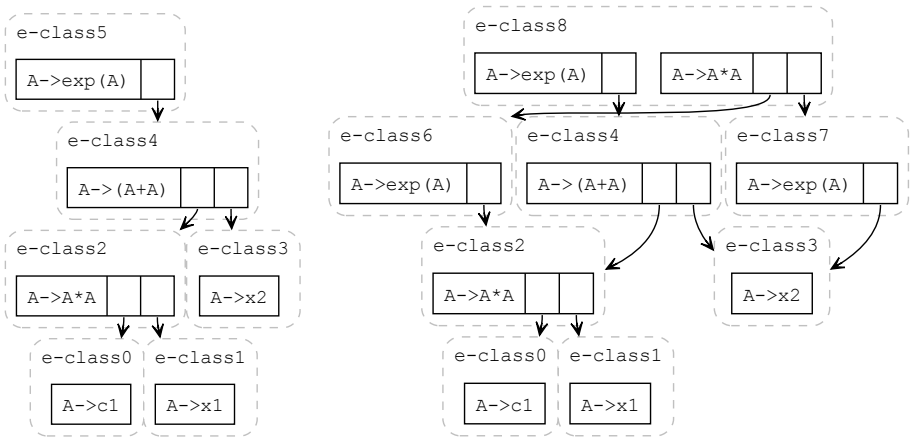


Figure 13: **(Left)** Initialized e-graph for expression $\exp(c_1 x_1 + x_2)$. **(Right)** The saturated e-graphs after applying one rewrite rule $\exp(a+b) \rightsquigarrow \exp(a)\exp(b)$.

E-graph for $\sin, \cos$ **operators.** Figure 14 demonstrates the application of the trigonometric identity $\sin(a+b) \rightsquigarrow \sin a \cos b + \sin b \cos a$ to the input expression $\sin(x_1 + x_2)$. The input expression is represented by the sequence of production rules $(A \to \sin(A),\ A \to A + A,\ A \to x_1,\ A \to x_2)$.

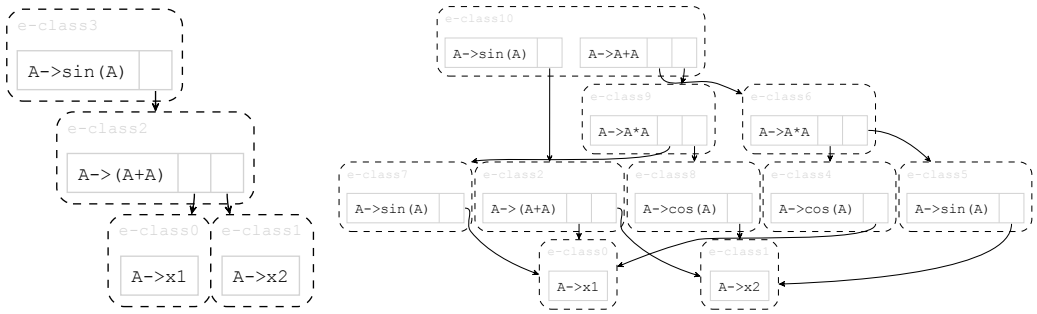


Figure 14: Applying the rewrite rule $\sin(a+b) \rightsquigarrow \sin a \cos b + \sin b \cos a$ in an e-graph representing expression $\sin(x_1 + x_2)$. **Left:** Initialized e-graph. **Right:** e-graph after applying the rewrite rule.

E-graph for $\partial/\partial x_i$ **operator.** Figure 15 shows the application of the partial derivative commutativity rule $\frac{\partial^2 f}{\partial x_i \partial x_j} \rightsquigarrow \frac{\partial^2 f}{\partial x_j \partial x_i}$ to the input expression $\frac{\partial^2 (x_1 + x_2)}{\partial x_1 \partial x_2}$. The derivation is represented by the following production rules: $(A \to \partial(A)/\partial x_1,\ A \to \partial(A)/\partial x_2,\ A \to x_1 + x_2)$.

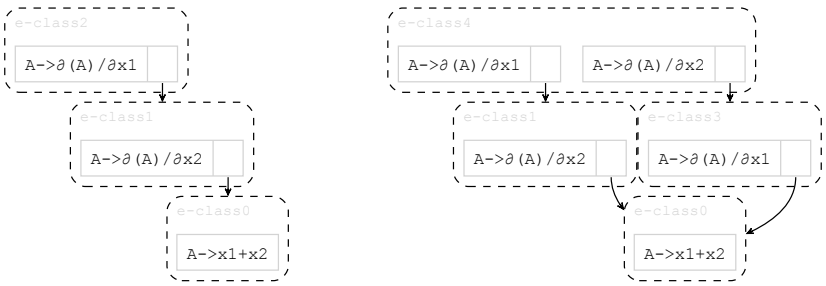


Figure 15: Applying the rewrite rule $\frac{\partial}{\partial x_i}\left(\frac{\partial f}{\partial x_j}\right) \rightsquigarrow \frac{\partial}{\partial x_j}\left(\frac{\partial f}{\partial x_i}\right)$ in an e-graph representing expression $\frac{\partial^2 (x_1 + x_2)}{\partial x_i \partial x_j}$. **Left:** Initialized e-graph. **Right:** e-graph after applying the rewrite rule.

E-graph visualization for case analysis in section 5.2.

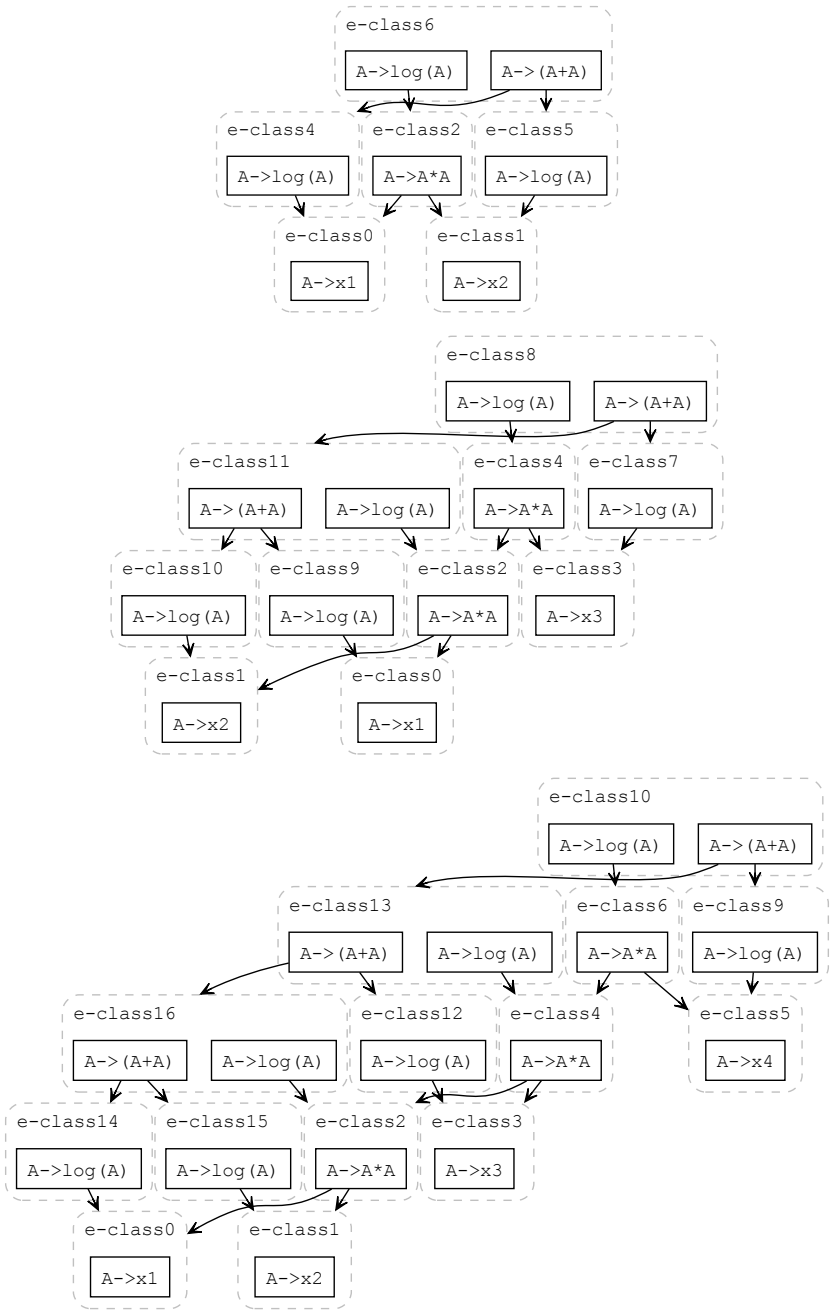


Figure 16: Visualization of e-graphs for expression $\log(x_1 \times \ldots \times x_n)$, using the rewrite rule $\log(ab) \rightsquigarrow \log a + \log b$. The three figures correspond to $n = 2, 3, 4$ accordingly.

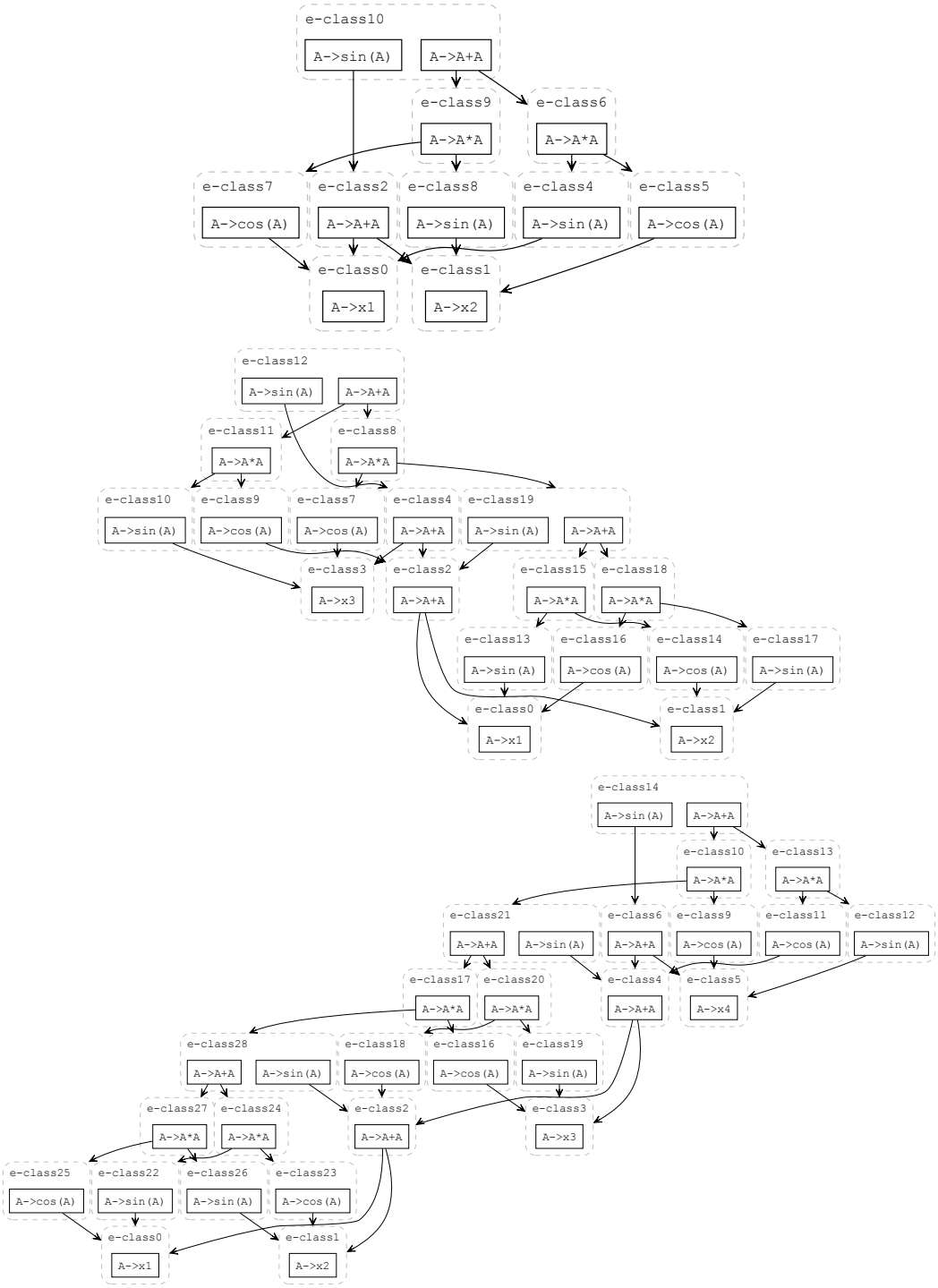


Figure 17: Visualization of e-graphs for expression $\sin(x_1 + \ldots + x_n)$, using the trigonometric rewrite rule $\sin(a+b) \rightsquigarrow \sin(a)\cos(b) + \sin(b)\cos(a)$. The three figures correspond to $n = 2, 3, 4$.

F.2 Additional Visualization for Selected Expressions in Feynman Dataset

E-graph for equation ID I.15.3x in the Feynman dataset. Figure 18 illustrates the application of the rewrite rules $\sqrt{ab} \rightsquigarrow \sqrt{a}\sqrt{b}$ and $a^2 - b^2 = (a + b)(a - b)$ to the input expression $\frac{x_0 - x_1 x_2}{\sqrt{c_1^2 - x_1^2}}$.

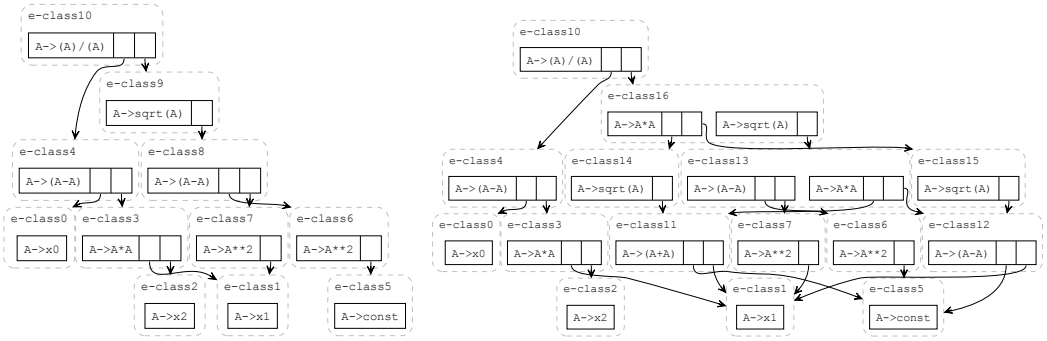


Figure 18: **(Left)** Initialized e-graph for the expression $(x_0 - x_1 x_2)/\sqrt{c_1^2 - x_1^2}$, where the equation ID is I.15.3x in the Feynman dataset. **(Right)** Saturated e-graph after applying the rewrite rule $\sqrt{ab} \rightsquigarrow \sqrt{a}\sqrt{b}$ and $a^2 - b^2 = (a + b)(a - b)$.

E-graph for equation ID I.30.3 in the Feynman dataset. Figure 19 illustrates the application of the rewrite rule $a^2/b^2 \rightsquigarrow (a/b)^2$ to the input expression $x_0 \frac{\sin^2(x_1 x_2/2)}{\sin^2(x_2/2)}$.

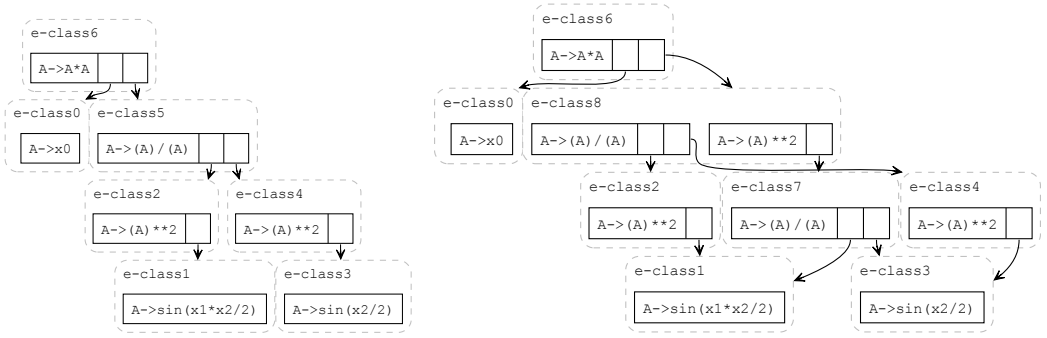


Figure 19: **(Left)** Initialized e-graph for the expression $x_0 \frac{\sin^2(x_1 x_2/2)}{\sin^2(x_2/2)}$, where the equation ID is I.30.3 in the Feynman dataset. **(Right)** Saturated e-graph after applying the rewrite rule $\log(a/b) \rightsquigarrow \log a - \log b$.

E-graph for equation ID I.44.4 in the Feynman dataset. Figure 20 illustrates the application of the rewrite rule $\log(a/b) \rightsquigarrow \log a - \log b$ to the input expression $c_1 x_0 x_1 \log(x_2/x_3)$.

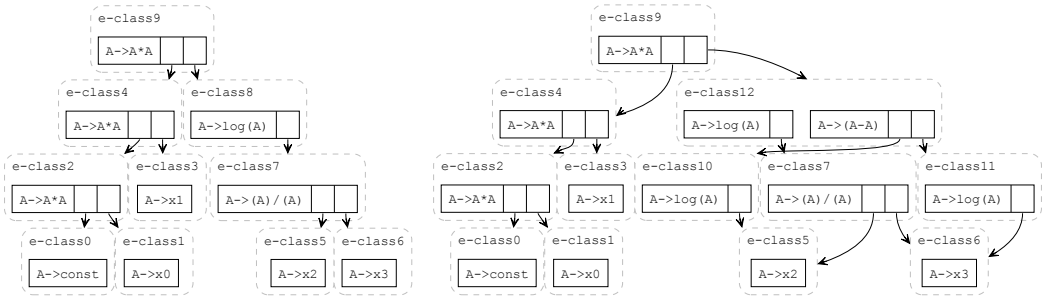


Figure 20: **(Left)** Initialized e-graph for the expression $c_1 x_0 x_1 \log(x_2/x_3)$, where the equation ID is I.44.4 in the Feynman dataset. **(Right)** Saturated e-graph after applying the rewrite rule $\log(a/b) \rightsquigarrow \log a - \log b$.

E-graph for equation ID I.50.26 in the Feynman dataset. Figure 21 illustrates the application of the rewrite rule $ab + ac \rightsquigarrow a(b + c)$ to the input expression $x_0(\cos(x_1 x_2) + x_3 \cos^2(x_1 x_2))$.

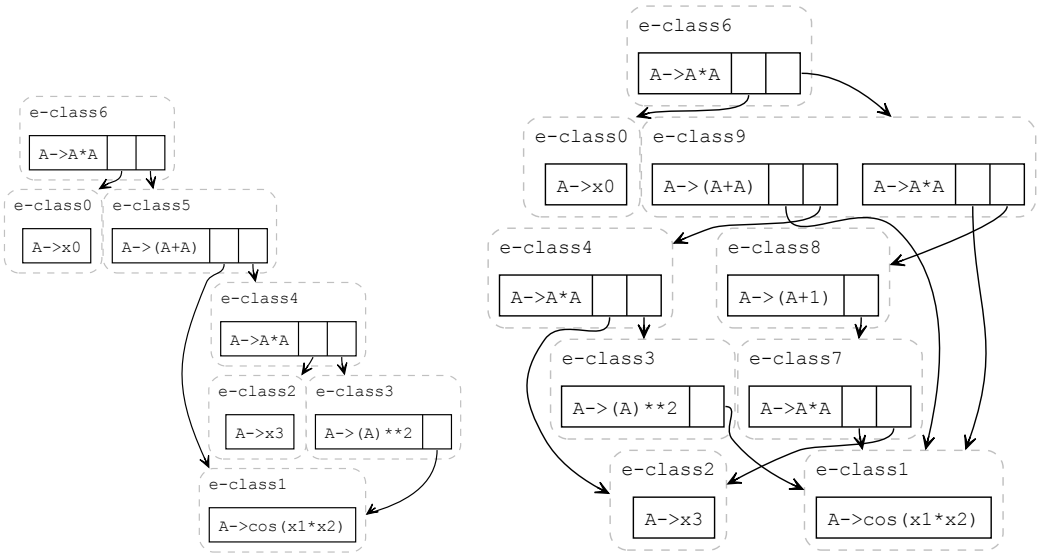


Figure 21: **(Left)** Initialized e-graph for the expression $x_0(\cos(x_1 x_2) + x_3 \cos^2(x_1 x_2))$, where the equation ID is I.50.26 in the Feynman dataset. **(Right)** Saturated e-graph after applying the rewrite rule $a + ba^2 \rightsquigarrow a(ba + 1)$.

E-graph for equation ID II.6.15b in the Feynman dataset. Figure 22 illustrates the application of the rewrite rule $\cos(a)\sin(a) \rightsquigarrow \frac{\sin(2a)}{2}$ to the input expression $\frac{x_0}{\exp(x_1 x_2/x_3)+\exp(-x_1 x_2/x_3)}$.

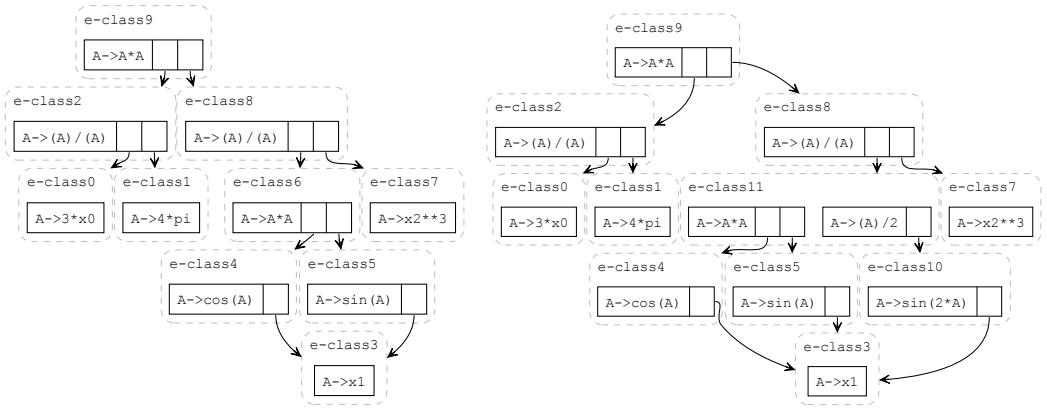


Figure 22: **(Left)** Initialized e-graph for the expression $\frac{3x_0}{4\pi}\frac{\cos x_1 \sin x_1}{x_2^3}$, where the equation ID is II.6.15b in the Feynman dataset. **(Right)** Saturated e-graph after applying the rewrite rule $\cos(a)\sin(a) \rightsquigarrow \frac{\sin(2a)}{2}$.

E-graph for equation ID II.35.18 in the Feynman dataset. Figure 23 illustrates the application of the rewrite rule $(\exp(a) + \exp(-a))/2 \rightsquigarrow \cosh a$ to the input expression $\frac{x_0}{\exp(x_1x_2/x_3)+\exp(-x_1x_2/x_3)}$.

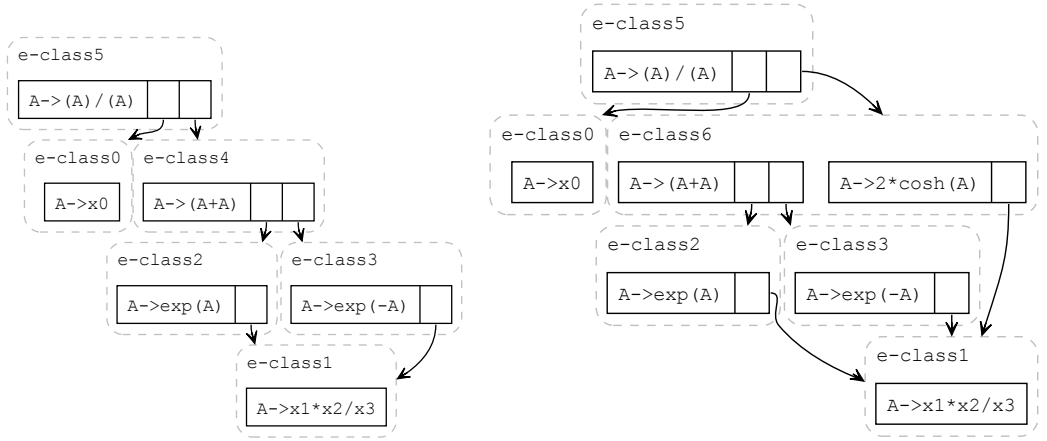


Figure 23: **(Left)** Initialized e-graph for the expression $\frac{x_0}{\exp(x_1x_2/x_3)+\exp(-x_1x_2/x_3)}$, where the equation ID is II.35.18 in the Feynman dataset. **(Right)** Saturated e-graph after applying the rewrite rule $(\exp(a) + \exp(-a)) \rightsquigarrow 2\cosh a$.

E-graph for equation ID II.35.21 in the Feynman dataset. Figure 24 illustrates the application of the rewrite rule $\tanh a \rightsquigarrow \frac{\exp(2a)-1}{\exp(2a)+1}$ to the input expression $x_0x_1\tanh(x_1x_2/x_3)$.

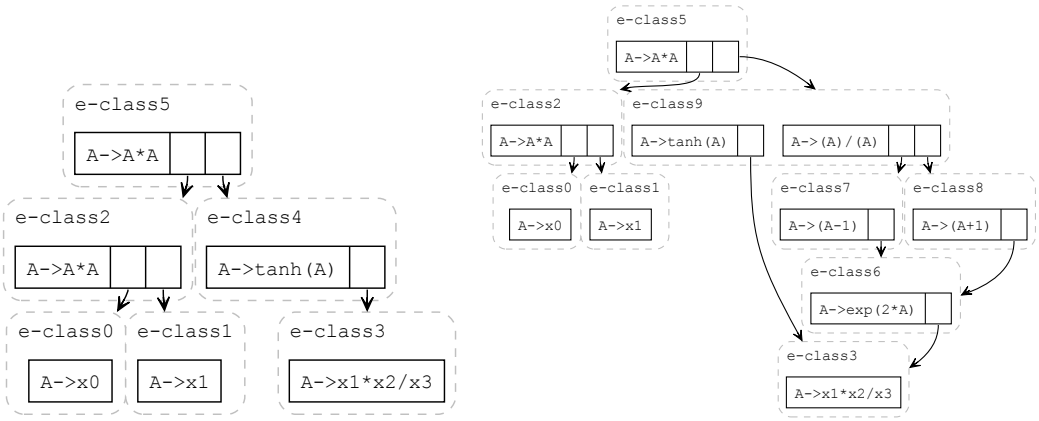


Figure 24: **(Left)** Initialized e-graph for the expression $x_0 x_1 \tanh(x_1 x_2 / x_3)$, where the equation ID is II.35.21 in the Feynman dataset. **(Right)** Saturated e-graph after applying the rewrite rule $\tanh a \rightsquigarrow \frac{\exp(2a)-1}{\exp(2a)+1}$.

F.3 Impact of Rewrite Rules

The runtime of EGG depends on the size of the rewrite-rule set. As more rules are included, the e-graphs maintain increasingly large sets of equivalent expressions.