Reinforcement Learning for Symbolic Equation Solving

Anonymous authors
Paper under double-blind review

Abstract

We present, to our knowledge, the first reinforcement-learning agent that solves *nonlinear* and *open* symbolic equations step-by-step from reward alone – with no supervised solution traces. Prior RL for equation solving is limited to linear equations (Poesia et al., 2021; Dabelow & Ueda, 2024), while one-shot neural solvers (Lample & Charton, 2020) require $\sim 10^8$ supervised pairs and emit an answer rather than a verifiable procedure. We cast algebra as an MDP with a *dynamic, per-state action space*, solved by a tree-structured policy network (**TreeMLP**), and reach *open* equations – those requiring a change of variables (CoV), e.g. completing the square – by exposing CoV as a macroaction whose *trigger* the agent learns; the substitution itself comes from a supervised generator interchangeable with a computer-algebra call (≤ 0.02 end-task effect), so the learned component is the trigger, not the substitution. The agent solves quadratic, cubic, quartic, and exponential open families (5-seed beam-decoded test accuracy **0.79** vs. 0.31 baseline; greedy ≈ 0.24, so the result is decode-time search informed by the policy) and matches the prior best on closed Common-Core equations (0.93 vs. ConPoLe's 0.925). At $10\times$ scale we additionally characterize – and partially mitigate with a learning-progress curriculum – a seed-level bimodality in which an otherwise-identical run either learns the task or collapses into a class-level policy trap.

1 Introduction

Most machine-learning approaches to symbolic mathematics map a problem directly to its answer Lample & Charton (2020): a model emits a solution in one shot. Many downstream goals, however, need the *procedure*, not just the answer – a sequence of legal, individually-checkable transformations that reaches the solution and exposes the trace and the policy that produced it. Differential-equation reductions, theorem-prover tactics, and didactic step-by-step solvers all consume such traces, and where no one-shot solver exists the procedure is the only thing on offer. We show that reinforcement learning can acquire this procedural capability for algebraic equation solving – and, for the first time, beyond the linear case to which prior RL is confined (Poesia et al., 2021; Dabelow & Ueda, 2024): our agent handles *nonlinear* (radical, exponential, trigonometric) equations and the harder *open* setting, where the solution requires introducing new structure via a change of variables. Crucially, it learns this from reward alone, with no supervised solution traces – unlike one-shot neural solvers, which translate a problem to its answer from $\sim 10^8$ supervised pairs (Lample & Charton, 2020) and expose no verifiable procedure. We choose algebra precisely because every step is checkable against ground truth, isolating the procedural-learning question from the question of *whether* a solution exists (a computer algebra system finds one in milliseconds). The setting is non-trivial: the state space is combinatorially large, the action space is *dynamic* (operations paired with sub-expressions extracted live from the current equation tree, so the legal-action set changes every step), and the solved signal is sparse, shaped by a dense complexity-delta reward (defined in Section 3).

We formulate symbolic algebra as an MDP whose actions pair an algebraic operation with a sub-expression of the current equation, the legal-action set re-derived from the state each step. The policy/value network is **TreeMLP**: node IDs are embedded, K rounds of strictly-local sum-aggregation message passing update node states, and a masked pool yields a fixed-size graph feature for the PPO heads. Three light inductive biases (curriculum, success-replay buffer, constant-relabel macroaction) and a value-guided beam decoder are layered on top; on CommonCore this is comparable to the prior best (ConPoLe; Poesia et al. 2021).

Our central new setting is *open* equation solving – equations whose solution requires introducing new structure via a CoV, such as completing the square. We expose CoV as a single *macroaction* in the same dynamic action space, its substitution supplied by a supervised generator π_{cov}; what the main policy learns is a CoV-*trigger* – *when* to invoke a CoV and how to compose it with ordinary steps (5-seed mean beam-decoded $test_{\text{beam}} = \mathbf{0.79}$ on `open_small`, 2.5× baseline; greedy mean ≈ 0.24, so this is decode-time search informed by the policy). The generator π_{cov} is interchangeable with SymPy's exact depression (swapping it moves end-task accuracy by $\leq$0.02), so the learned contribution is the trigger, not the substitution. Finally, scaling the open benchmark 10× exposes a sharp *seed-level bimodality* – an otherwise-identical run either learns the task or collapses into a class-level policy trap – which we characterise and partially mitigate with a learning-progress curriculum (Section 4.2.3).

Contributions. (a) *The first reward-driven RL agent for nonlinear and open symbolic equation solving.* We give an MDP formulation with a dynamic per-state action space that solves nonlinear equations step-by-step, and a *learned change-of-variables trigger macroaction* that extends it to *open* equations – composable with closed-equation steps under a single policy. Both go beyond prior RL, which is confined to linear equations (Poesia et al., 2021; Dabelow & Ueda, 2024); the agent learns from reward alone (the substitution generator π_{cov} is supervised but interchangeable with a CAS, so the learned contribution is the trigger, not the generator). (b) *TreeMLP and supporting machinery*: a tree-structured policy/value backbone for symbolic state, together with success-replay, value-guided beam, and anti-loop reward shaping – which together match the prior best on closed CommonCore equations (0.93 vs. ConPoLe's 0.925). (c) *A scale-induced failure mode and candidate mitigation*: at 10× scale we additionally report a sharp *seed-level bimodality* (a class-level policy trap; probes find no evidence for capacity loss, data skew, or representation collapse, though these checks are necessary-not-sufficient) and partially reduce it with a UCB learning-progress curriculum – a promising trend rather than a powered fix. We study a CAS-solvable domain so that every step is checkable; the contribution is the learned *procedure* – a verifiable trace and the policy that produced it – not the answer a CAS already provides.

2 Related work

RL for symbolic equation solving. The closest prior work treats symbolic algebra as a sequential-decision problem with exact symbolic transitions. Poesia et al. (2021) introduce ConPoLe, a contrastive learner whose CommonCore-equations benchmark we adopt; ConPoLe's equation domain consists of linear equations under low-level axioms, on which it reaches 92.5% accuracy. Dabelow & Ueda (2024) train an RL agent on a stack-calculator interface for linear-equation manipulation. We move beyond linear equations to nonlinear (radical, exponential, trigonometric) instances and to the harder *open* setting, where the solution requires generating new structure via a change-of-variables. To our knowledge, prior RL work in this space has focused on linear-equation manipulation rather than the open-equation setting we consider here.

Neural symbolic math. Lample & Charton (2020) train a seq2seq Transformer to map equations to closed-form solutions for integration and ODEs, treating symbolic mathematics as a translation task. Their setting differs from ours in two ways: (i) the model produces a one-shot solution rather than a verifiable step-by-step trace, and (ii) training is fully supervised on $\sim 10^8$ (equation, solution) pairs synthesised from rule-based generators. We learn from a per-step complexity-delta reward (defined in Section 3) plus a terminal-solved bonus, rather than from supervised full solution traces; the only supervised component is the substitution generator π_{cov}, which is trained on $\sim 10^3$ known target substitutions.

Symbolic regression and program synthesis. Deep Symbolic Regression / DSO Petersen et al. (2021) uses RL to generate symbolic expressions that fit numeric data; DreamCoder Ellis et al. (2021) synthesises programs by iteratively expanding a library of learned subroutines. Both *generate* formulae; we instead manipulate a given equation by legal transformations. Closest to our setting, Li et al. (2022) learn high-level actions for step-by-step mathematical reasoning (including equation solving) by inducing reusable abstractions from an agent's own solutions via expert iteration; our CoV macroaction is similar in spirit (and to DreamCoder's (Ellis et al., 2021) library expansion), but our unit of composition is a single concrete substitution – the CoV trigger – rather than a library of induced abstractions.

Graph encoders for symbolic state. Standard graph encoders – GCN Kipf & Welling (2017), GraphSAGE Hamilton et al. (2017) – assume locally-similar features and large undirected neighbourhoods; symbolic expression trees satisfy neither (small ordered vocabulary, sparse rooted DAGs). TreeMLP (Section 3.1) drops their degree normalisation and learned edge transforms for a strictly local sum-aggregation update aligned with parent-to-child tree semantics, and – to our knowledge – has not been evaluated end-to-end as a PPO policy backbone for step-by-step symbolic equation solving.

Formal mathematical reasoning. Peano Poesia & Goodman (2023) formalises introductory algebra as theorem proving and finds existing RL insufficient without explicitly induced tactics; AlphaZero-style search has been applied to formal-theorem proving at much larger scale. These settings target proof-style validity rather than computational equation solving and are complementary to our work.

Curriculum learning and capability collapse. Matiisen et al. (2019) introduce a learning-progress (LP) curriculum in which a teacher samples tasks proportionally to their estimated marginal improvement; ACTOR-CURATOR Gu et al. (2026) extends this with an explicit policy-improvement bandit. Our UCB-LP variant (Section 4.2.3) replaces the vanilla-LP exploration floor with a UCB1 Auer et al. (2002) bonus on per-class sampling weights. Recent work also documents bimodal-across-seeds capability acquisition: *Random Scaling of Emergent Capabilities* Zhao et al. (2025) attributes such patterns to capacity thresholds; *Policy Confounding* Suau et al. (2024) and *Capability Boundary Collapse* Dong et al. (2025) name the lock-in mechanism for on-policy / sparse-reward agents. Our bimodality on `open_large` is the symbolic-RL instance of this family.

3 Problem Formulation

Our Markov Decision Process (MDP) formulation is

States: equations like $ax + b = 0$ or $cx + d = -x/b$. We vectorize these using preorder traversal over the equation's expression tree (Appendix A.12, Figure 6) and pad to a maximum length $L = 50$. Operations and symbols are mapped to integers, e.g. $\{\text{add}:1,\ \text{sub}:2,\ \text{mul}:3,\ \ldots,\ x:5,\ a:6,\ b:7,\ \ldots\}$; both lhs and rhs are encoded and concatenated. For example, $x + a \to [\text{add}, x, a] \to [1, 5, 6, \text{PAD}, \ldots]$.

Actions: represented as $(\textit{operation}, \textit{term})$ pairs, such as (sub, b) or (div, a). Formally,

$$O = \{\text{add},\ \text{sub},\ \text{mul},\ \text{div}\} \cup \{\text{square},\ \text{sqrt},\ \text{exp},\ \text{log},\ \text{sin},\ \text{cos},\ \text{asin},\ \text{acos}\}, \tag{1}$$

$$T = \text{SubExpr}(\text{lhs}) \cup \text{SubExpr}(\text{rhs}), \tag{2}$$

$$A = (O \times T) \cup \{(\text{expand}, \text{None}),\ (\text{collect}, x),\ (\text{mul}, -1)\}. \tag{3}$$

Notice the first set of operations take two arguments, the second one argument. For the terms, we choose the list of sub-expressions in the lhs and rhs expression trees. Example of sub-expressions are

$$ax + b = 0 \ \Rightarrow\ \{a, x, ax, b\} \tag{4}$$

$$\tfrac{ax+b}{cx+d} + e = 0 \ \Rightarrow\ \{a, x, ax, b, c, d, cx, cx + d, ax + b\} \tag{5}$$

This term set is expressive enough to solve rational equations and all other equation types we consider in this paper. It is also dynamic: the list of sub-expressions is derived from the equation/state and thus has variable length. Looping back to the operations, we also include $(\text{mul}, -1)$ and

$$\text{expand:} \qquad cx + d + e(ax + b) \to aex + be + cx + d$$

$$\text{collect } x\text{:} \qquad ax + bx \to (a + b)x$$

These allow the agent to perform key algebraic steps (Appendix A). Finally, we index A serially and cap it at $|A| = 50$ (the largest set any dataset equation produces is ~ 40 after dynamic expansion; varying $|A| \in \{40, 70, 100\}$ produced no material change in single-seed checks). Illegal actions (e.g. division by zero) are masked at each step (Appendix A.3).

Rewards. Define the *complexity* C of an equation as the total number of nodes and edges in the expression tree.[1] For example, $C(ax+b) = 5+4 = 9$ and $C(ax^2+bx+c) = 10+9 = 19$. The reward is then

$$R(\text{action}) = C(\text{equation}) - C(\text{equation after action}) \tag{6}$$

i.e., reward actions that simplify the equation.

State Transition Function. We use SymPy Meurer et al. (2017) to apply operations to a tracked (lhs, rhs) pair. For example, from $(ax+b,\, 0)$, (sub, b) gives $(ax, -b)$ and then (div, a) gives $(x, -b/a)$. The episode terminates when $lhs = x$ and substituting the rhs into the original equation simplifies to 0.

Closed vs. open equations. The MDP formulation above only handles equations that are *closed*, in the sense that solving them requires manipulating terms already present in the equation/sub-expression list. By contrast, solving *open* equations requires adding new, out-of-equation terms or clever substitutions. A classic example is the quadratic equation $ax^2+bx+c=0$ which is solved by completing the square – adding $(b/2a)^2$ to each side. We call this reasoning *out-of-term-set*, since the term $(b/2a)^2$ is *not* in the term set we have defined; "generative" here means out-of-term-set, not open-ended expression synthesis. Equations that require such out-of-term-set actions are beyond the scope of the closed-action formulation just described, and were not studied in prior RL work either Poesia et al. (2021); Dabelow & Ueda (2024). We extend the formulation to the open setting in Section 4.2 by exposing the change-of-variables as a learned macroaction inside the same dynamic action space.

3.1 TreeMLP: a tree-structured policy network

We introduce **TreeMLP**, a lightweight graph-based policy/value network purpose-built for symbolic expression trees. Nodes are tokens (operators, symbols, or terminals) and edges are parent→child relations. Each node ID is embedded and linearly projected to a hidden size, then K rounds of message passing update the node states with a strictly-local sum aggregator over incoming edges (no degree normalisation): at round t,

$$h_i^{(t+1)} = \text{MLP}\left(\left[\text{emb}_i \,\|\, \sum_{j \to i} h_j^{(t)} \right] \right), \tag{7}$$

where the MLP is two ReLU layers and emb_i is the static node embedding. Padding-aware masks zero out invalid nodes and edges at every stage. After K iterations a fixed-size graph feature is produced by masked mean (or max) pooling over valid nodes and fed to the PPO policy and value heads.

These choices – direct node-ID embedding, sum aggregation without degree normalisation, no learned edge transforms, masked pooling – align TreeMLP with the rooted, sparse structure of expression trees and avoid two encoder pathologies on this domain: GCN's Kipf & Welling (2017) degree normalisation dilutes a single parent's message when an operator has many children (e.g. add with 5 summands), and GraphSAGE's Hamilton et al. (2017) learned edge transforms overfit the small algebraic vocabulary. TreeMLP outperforms both at equal hidden size on the closed-equation small dataset (Appendix A.10; Figure 3) and trains stably with PPO.

The full architecture diagram, message-passing pseudocode, and ablations against vanilla MLP / GCN / GraphSAGE baselines are in Appendix A.10.

3.2 Inductive biases

On top of TreeMLP we use three additional inductive biases.

Curriculum learning. The agent solves a single equation chosen randomly from a larger set each episode. Equations are selected via *inverse sampling*: the probability of choosing an equation is inversely proportional to the number of times it has been solved before, so the agent spends more time on equations it has not yet mastered.

[1] We judged number of nodes+edges correlates better with algebraic complexity than number of nodes alone.

Success-replay buffer. Solution traces in our environment are *exact*: once a sequence of (op, term) actions solves an instance (e.g., $ax+b=0$), replaying the same actions deterministically reproduces the solution. We exploit this with a light-weight success-replay path: solution traces $(s_t, a_t)_t$ are stored to a memory buffer and periodically mixed into the on-policy rollouts.

Relabel-constants macroaction. To improve parameter sharing across structurally identical equations that differ only by symbol names, the agent has a *relabel-constants* action that applies a consistent partial renaming (e.g., $\{d \mapsto a,\ e \mapsto b,\ \dots\}$) to the current state, mapping arbitrary coefficient symbols onto a canonical alphabet $\{a, b, c\}$. The environment stores the substitution stack and, upon solving, unwinds it so the answer is expressed in the user's original symbols. This reduces input vocabulary entropy, prevents overfitting to symbol IDs, and makes action terms like "subtract b" reusable across many instances.

Curiosity bonuses (negative result). In single-seed runs the curiosity-based exploration bonuses (ICM Pathak et al. (2017), RND Burda et al. (2019), NGU Badia et al. (2020)) shown to help vanilla-MLP policies in prior work Dabelow & Ueda (2024); O'Keeffe (2025) *fail to match* the no-curiosity baseline on top of TreeMLP (on `abel_level3`, a closed-equation difficulty tier; see Section 4.2.1); this is suggestive, not conclusive. Details in Appendix A.12. A plausible explanation is that TreeMLP's structural inductive bias already supplies the exploration signal curiosity provides for an MLP.

4 Results

4.1 Closed equations

Algorithms and training. We use Stable-Baselines3's Raffin et al. (2021) PPO Schulman et al. (2017) (A2C/DQN were worse in initial experiments); hyperparameters are in Appendix A. Each run is 10^7 steps with evaluation every 5×10^5, over three random seeds.

Datasets. We construct two recursive-template datasets by starting from x and applying random (operation, term) actions, with terms restricted to $\{a, b, c\}$ (to favour long over wide equations). We discard equations SymPy cannot solve and count multi-root cases solved when any one root is found (e.g. $\sin x = 0$ at $x = 0$). `small` contains all depth-<4 equations (3874, subsampled $10^3/10^2$ train/test); `large` contains all depth-<5 ones (15625, subsampled $10^4/10^3$). We additionally use the CommonCore `poesia` benchmark.

Results. Table 1 reports end-of-training greedy test accuracy for the inductive-bias ablation. On the `poesia` CommonCore benchmark, PPO-TREE-RC-BUF reaches 0.93 greedy test accuracy on the full benchmark, matching ConPoLe's 0.925 Poesia et al. (2021) under matched greedy decoding. Restricting to non-degenerate templates (excluding edge cases the closed-action set cannot reach) raises accuracy further – to 0.955 for PPO-TREE-RC and ≥ 0.98 on the solvable subset (Appendix A.5). Each bias contributes: TreeMLP alone lifts `poesia` from 0.17 (vanilla MLP) to 0.80; relabel-constants adds a further jump on `small`; and the success-replay buffer is critical for scaling – PPO-TREE-RC-BUF degrades least with depth ($0.85 \to 0.50$ `small`$\to$`large`, vs $0.85 \to 0.10$ for PPO-TREE-RC).

Table 1: Greedy test accuracy on the three closed-equation benchmarks for the inductive-bias ablation. ConPoLe / ConPoLe-local numbers from Poesia et al. (2021). `small`/`large` are our depth-<4 / depth-<5 internal benchmarks; `poesia` is the CommonCore equations split of Poesia et al. (2021).

Algorithm	small	large	poesia
ConPoLe-local	n/a	n/a	0.765
ConPoLe	n/a	n/a	0.925
ppo	0.04	0.02	0.17
ppo-tree	0.25	0.10	0.80
ppo-tree-rc	0.85	0.10	0.85
ppo-tree-buf	0.55	0.40	0.85
ppo-tree-rc-buf	**0.85**	**0.50**	**0.93**

Learned representations. t-SNE van der Maaten & Hinton (2008) projections of the TreeMLP pooled embeddings cluster cleanly by equation family (linear, radical, trig, log occupy disjoint regions), indicating the encoder captures structural identity rather than surface tokens (Appendix A.12, Figure 7).

4.2 Open equations

We now consider the harder class of *open equations*: those that cannot be solved by rearranging the subexpressions already present in the equation, but require introducing a new term via a *change of variables* (CoV) – a substitution $x = \phi(y)$ that re-expresses the equation in a new variable y. We consider four template families, each paired with its solving substitution:

$$ax^2 + bx + c = 0, \quad x = y - b/(2a) \tag{8}$$

$$ax^3 + bx^2 + cx + d = 0, \quad x = y - b/(3a) \tag{9}$$

$$ax^4 + bx^3 + cx^2 + dx + e = 0, \quad x = y - b/(4a) \tag{10}$$

$$ae^{kx} + be^{-kx} + c = 0, \quad x = \tfrac{1}{k}\log y \tag{11}$$

Each substitution depresses (or, for the exponential, rationalizes) the equation, after which standard closed-action steps finish the solution. The polynomial families are restricted to fully-depressable special forms: e.g. the quartic dataset constrains (c, d, e) to functions of (a, b) so that $y = x + b/(4a)$ reduces the equation to $Ay^4 + B = 0$, solved by two applications of $\sqrt{\cdot}$; generic quartics, which admit no closed-form depression by a fixed CoV, are excluded. The exponential case yields a Laurent polynomial in y that resolves to a quadratic after one further CoV (exponential block of Table 8). We expose the CoV as a single *macroaction* in the same dynamic action space: when the policy selects it, the substitution ϕ is generated by a learned policy π_{cov} (Section 4.2.4) and applied to the equation. The main policy thus learns *when* to invoke a CoV and how to compose it with ordinary steps, while π_{cov} supplies the substitution – keeping the open problem inside the same MDP, with one policy trained on a mixed closed/open dataset (Section 4.2.1).

Benchmark scope. We call this the *Restricted-Open* setting: arbitrary same-degree polynomials with non-special coefficients (e.g. a generic $ax^4 + bx^3 + cx^2 + dx + e$) are out of scope, as they admit no closed-form depression by a fixed CoV. The goal is to study whether RL can learn *when* and *how* to invoke a CoV macroaction in a controlled regime where one exists; extending to fully open families is future work.

4.2.1 Method

We train a single TreeMLP-PPO policy on mixed datasets of closed and open equations: `open_small` (916 train / 91 test, stratified across the four CoV families and the `abel_level1/2/3` closed classes) and a 10×-larger `open_large`. (`abel_level1/2/3` are three closed-equation difficulty tiers of increasing tree depth and coefficient structure, `abel_level3` the hardest; throughout, *coverage* denotes the fraction of *training*-set equations the agent solves, as distinct from held-out test accuracy.) The policy carries over the closed-equation inductive biases – curriculum learning, constant relabeling, and a success-replay buffer (Section 4.1) – with four additions specific to the open setting, which we state up front and ablate below.

Action set additions. The open setting adds (i) a cube-root primitive (needed because the depressed-cubic CoV collapses an equation to $ay^3 + k = 0$ which $\sqrt{\cdot}$ alone cannot finish) and (ii) the CoV macroaction itself, with a per-episode cap of 3 applications to permit nested substitutions (an exponential equation first becomes a rational form, then a quadratic).

Action-diversity penalty. A small reward penalty ($\alpha = 0.1$) is subtracted whenever the current action's integer index matches the previous step's – exact (operation, term) identity, not operation alone. The penalty targets short-cycle loops: 93% of unsolved `abel_level3` failures on the larger benchmark (Section 4.2.3) are 1-cycles (81%, caught by the penalty) or $a \to b \to a$ 2-cycles (12%, slipping past a one-step penalty; we return to this in Section 6).

Freshness-managed success replay. The buffer discards its oldest half every 20 rollouts; a flat FIFO buffer goes stale (the policy memorises its own past successes), whereas refreshing keeps the behaviour-cloning target on the current frontier and improves sample efficiency $\sim 5\times$.

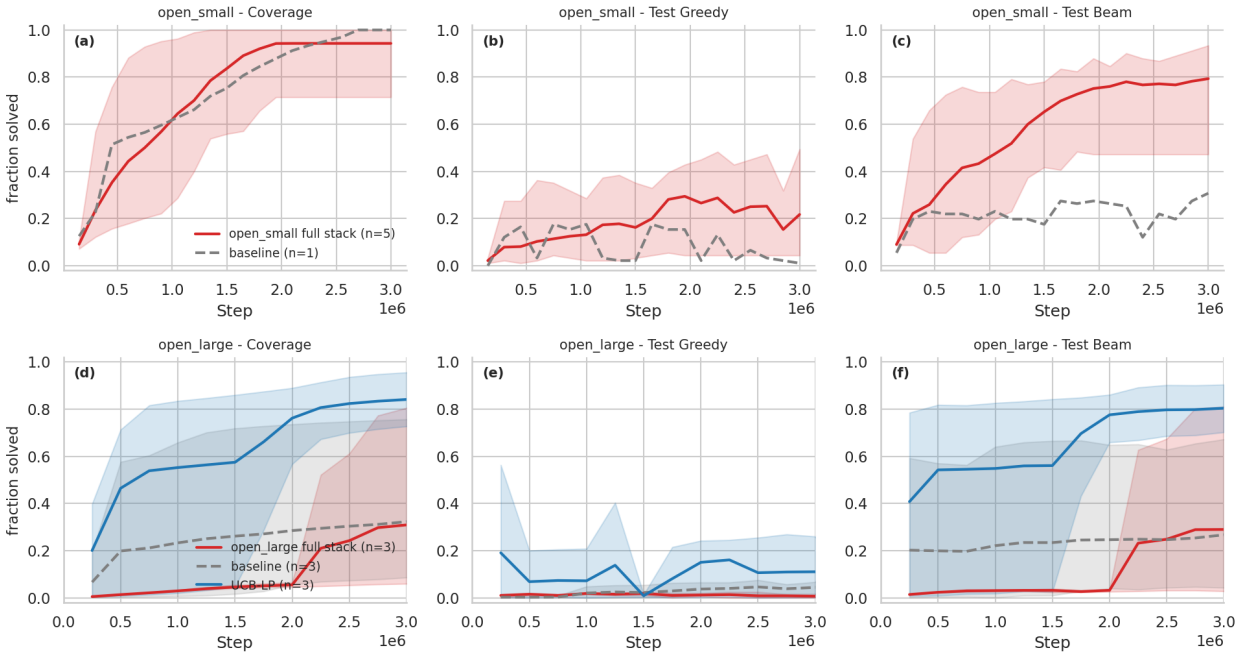

Figure 1: **Open-equation learning curves** (coverage, greedy and beam-width-5 test accuracy) for `open_small` (top) and `open_large` (bottom). Red: full method stack; grey dashed: baseline; blue: UCB-LP. Solid lines are seed means, bands min/max; seed counts per cohort are given in the text. Per-seed distributions are in Figure 2.

Value-guided beam search. At evaluation we score beam paths by

$$\text{score}(\text{path}) = \sum_t \log \pi_\theta(a_t \mid s_t) + \lambda V_\theta(s_T) - \beta C(s_T), \tag{12}$$

combining the trained policy log-prob with a value-head look-ahead (V_θ) and an equation-complexity proxy (C). We use width 5 and $(\lambda, \beta) = (1, 0)$ throughout ($\lambda = \beta = 0$ recovers plain beam); varying widths $\{3, 5, 7, 9\}$ and $\lambda \in \{0, 1\}$ produced no material change at convergence (observed spread ≤ 0.025, no table). Beam entries reaching the same canonical equation are deduplicated.

4.2.2 Results

A plain PPO-TREE-RC-BUF-COV agent on `open_small` reaches *coverage* = 1.0 on the training set but only $test_{\text{beam}} = 0.31$ on test (`open_small`, $n = 1$): it memorizes a single CoV recipe and fails to generalize across the four families. Adding the four open-equation ingredients above and selecting the best checkpoint by validation accuracy (Section A.13), we obtain a 5-seed mean $test_{\text{beam}} = \mathbf{0.79}$ (`open_small`, $n = 5$, range 0.47–0.93) – a 2.5× lift over the baseline. This is a decode-time *search* (beam width 5) result; the corresponding 5-seed greedy mean is only ≈ 0.24. One of the five seeds early-stopped at 1.95M of 3M steps before fully learning the CoV classes ($test_{\text{beam}} = 0.47$), and is included in the 5-seed mean as reported. Figure 1 (top row) shows the learning curves with mean and min-to-max band across the five seeds.

Ablation. Table 2 shows the effect of cumulatively adding each ingredient. Rows 1–4 are illustrative *single-seed* diagnostics: their magnitudes are not seed-controlled and, given the seed-level variance documented in Section 4.2.3, should not be read as effect sizes. The final step (0.62 → 0.79) additionally compares a single-seed configuration against the 5-seed full-stack mean, so it is not a like-for-like increment.

Per-class accuracy. On `open_small` the full stack solves all four CoV classes at 88–100%, lifting the exponential class from 0% to 88%; `abel_level3` remains weakest (0.53, full stack, 4-seed mean), its failures dominated by short action cycles (breakdown in Table 9). On `open_large` the pattern is sharper: across

configuration	$test_{\text{beam}}$
baseline (PPO-TREE-RC-BUF-COV), plain beam	0.31 (n=1)
+ action-diversity penalty	0.34 (n=1)
+ value-guided beam & dedup	0.40 (n=1)
+ cube-root & nested CoV (flat buffer)	0.62 (n=1)
+ freshness-managed buffer (**full**)	**0.79** (n=5)

Table 2: **Cumulative configuration table** on `open_small` ($test_{\text{beam}}$, beam width 5). Each row adds one ingredient to the configuration above it. Rows 1–4 are *illustrative single-seed diagnostics*; magnitudes are not seed-controlled and should not be read as effect sizes (cf. the seed-level variance of Section 4.2.3). The bottom row is the 5-seed mean reported in the main result, so the final step compares a single-seed config to a 5-seed mean (not like-for-like). We therefore report configurations rather than "$+0.XX$ contributions".

escape seeds the CoV classes saturate (~ 0.98), `abel_level3` stays weak (escape-seed mean 0.39, range $[0.07, 0.55]$; Table 10).

Solution trace: nested change-of-variables. The exponential class is the only one of the four CoV families that requires *two* CoV steps; the agent discovers this nested recipe on its own (full greedy trace in the exponential block of Table 8, Appendix A.8). The policy first applies CoV ($x \to \log x/k$) to turn the exponential equation into a rational one, expands to a quadratic, then invokes CoV *again* ($x \to x - B/(2A)$) to depress the quadratic. The same agent solves quadratic, cubic, and quartic classes with a single-CoV recipe.

Search informed by the policy. The greedy/beam gap is large (`open_small` 5-seed final-step greedy mean 0.24 vs. beam 0.79), so the policy represents solution paths even when its top-1 action is unreliable. Non-learned search over the same action space lags: on `open_small`, BFS reaches 0.26 ($N = 10^5$ expansions/eqn) and a best-first / A* search guided by the *same* complexity heuristic the reward uses reaches 0.38–0.64, both short of the agent's 0.79 `open_small` beam (5-seed mean) at $\sim$7$\times$ less per-equation search. The learned policy provides informed pruning of the search tree, not the existence of a solution in it.

Decomposing the open-equation pipeline. We re-evaluate the converged `open_large` agent (seed109, `open_large`, $n = 1$; $test_{\text{beam}} = 0.81$ on the full 1,001-equation held-out file) in three configurations (Table 3): (A) the full learned pipeline; (B) the CoV macroaction *disabled* (the agent must solve using only closed-action steps); and (C) an *oracle CoV* (SymPy's depression substitution, applied on 818 of the 1001 CoV-family episodes; closed-class equations unaffected). All denominators are the full 1001-equation held-out file. Configuration B does *not* show CoV is a train-from-scratch necessity: the trained policy simply does not degrade gracefully when CoV is masked at test time, a regime it was never trained for. Configuration C isolates substitution *generation*: swapping π_{cov} for SymPy's exact depression moves accuracy by ≤ 0.02 across escape seeds (the decomposition seed `seed109`: $0.81 \to 0.798$; three further escape seeds, $A \to C$: $0.79 \to 0.80$, $0.90 \to 0.90$, $0.91 \to 0.90$), so the open-equation accuracy comes from the CoV-trigger and closed-action policy, not π_{cov}'s decoder. Decoupling the learned trigger from the learned substitution is a natural further decomposition.

Table 3: **Pipeline decomposition** on `open_large` (seed109; beam width 5; denominator = 1001 held-out equations). Config B masks the CoV macroaction at test time; config C substitutes SymPy's exact depression for π_{cov} on the 818 CoV-family episodes.

config	what changes	beam acc
A	full learned pipeline	0.81
B	CoV macroaction masked at test	0.004 (4/1001)
C	oracle (SymPy) CoV vs. π_{cov}	0.798 (799/1001)

Validation-based checkpoint selection. We split each test set into a 30% validation / 70% held-out test slice (seeded shuffle, identical across cohort seeds), pick each seed's checkpoint by val beam, and report test

on the disjoint slice. For `open_small`'s 91-equation test set this is 27 validation / 64 test ($\lfloor 0.30 \cdot 91 \rfloor = 27$); all `open_small` test-beam numbers in this paper are on the 64-equation test slice. On `open_small` the val-best $test_{\text{beam}}$ is within ~ 0.02 of the final-step value (monotone-to-plateau learning, no peak-and-decay), ruling out test-set tuning.

4.2.3 Scaling to `open_large`: a bimodal failure mode and a candidate mitigation

The bimodality. On the 10×-larger `open_large` benchmark (10,000 train / 1,001 held-out equations, same composition, split 30/70 into validation/test), the agent exhibits a sharp seed-level *bimodality*. The three headline cohorts alone comprise $5 + 3 + 8 = 16$ training runs (baseline, vanilla LP, UCB-LP; Table 4); the full intervention sweep additionally screens five non-headline cohorts at 3 seeds each (Table 11). Across these runs, results partition cleanly into two regimes separated by a wide empirical gap – no seed lands in $(0.10, 0.27)$: an *escape* regime with $test_{\text{beam}} \geq 0.2$ (observed range $[0.27, 0.91]$) and a *stall* regime with $test_{\text{beam}} \leq 0.10$. We use $beam \geq 0.2$ as the escape threshold throughout; because no seed lands in the empirical gap $(0.10, 0.27)$, the escape counts are invariant to any threshold in that interval, so the binary escape/stall classification is robust to threshold choice (though not to seed sampling). The same code, data, and hyperparameters produce both regimes; only the seed differs. This resembles bimodal-across-seeds capability acquisition in LM pretraining Zhao et al. (2025) and the *policy confounding* Suau et al. (2024) / *capability boundary collapse* Dong et al. (2025) failure modes.

Mechanism: per-class trap dynamics. A post-hoc analysis of in-training solve events classifies each successful step by the source class of the equation it solved. Escape seeds distribute their solves roughly uniformly across the four CoV classes (20–25% each) and `abel_level3` ($\sim 10\%$). Stall seeds concentrate $\geq 85\%$ of their solves on `abel_level3`, the easiest class *during training*, and approach zero on the CoV classes. Stalled seeds *did* learn – they routinely solve training `abel_level3` instances – but fall into short action cycles on held-out instances, so even their held-out `abel_level3` accuracy is low: the training-easy/test-hard gap is itself a symptom of memorising particular training equations rather than a transferable procedure. **Escape correlates with class diversity in training events; stall corresponds to `abel_level3` lock-in**, via the short-cycle failure pattern quantified in Section 4.2.1 (93% of unsolved cases are 1- or 2-cycles).

Across three probes we find *no evidence* for the three most natural alternative explanations: *capacity loss / dormant neurons* Sokar et al. (2023) (0% dormant on both stuck and escape seeds at the standard ReDo threshold), *data-distribution skew* (an `abel_level3`-skewed training set transfers as 0.04 to the standard split), and *encoder representation collapse* (t-SNE clusters cleanly on stalled seeds). Details in Appendix A.12. These probes are necessary-not-sufficient: in particular, the failure is localized to the policy/value head, which the t-SNE *encoder* probe does not interrogate, so a clean encoder does not rule out a head-level representation issue. With that caveat, the evidence is most consistent with a *policy-attractor* phenomenon rather than a capacity, data, or input-representation failure.

Intervention sweep. We evaluated seven intervention cohorts targeting the failure mode; Table 4 reports the headline cohorts (baseline, vanilla LP curriculum, UCB-LP) with the full 9-row sweep in Appendix A.12. The most effective is a *UCB-style learning-progress curriculum* (UCB-LP), which replaces vanilla LP's Matiisen et al. (2019) heuristic exploration floor with a UCB1 Auer et al. (2002) bonus on per-class sampling weights: $w_c \propto \hat{p}_c + \beta\sqrt{\log T / n_c}$ ($\hat{p}_c$ = EMA learning progress on class c, n_c its visit count, $\beta = 2.0$), structurally similar to ACTOR-CURATOR Gu et al. (2026). Across $n = 8$ training runs it achieves **6/8** escape with mean $beam$ = **0.83** over escape seeds (range 0.69–0.91); the best UCB-LP seed reaches 0.911 (+0.10 over the previous single-seed best of 0.81). The escape-rate change from 3/5 (baseline) to 6/8 (UCB-LP) is *not* statistically significant (two-sided Fisher exact $p \approx 1.0$; Wilson 95% CIs $3/5 \approx [0.23, 0.88]$, $6/8 \approx [0.41, 0.93]$); a powered claim would need roughly $\geq$20 seeds per cohort. UCB-LP should therefore be read as a promising/trending direction, not a demonstrated fix. The two non-escape seeds were already in the stall regime ($beam \leq 0.05$) before crashing on a SymPy/mpmath `OverflowError` (a pipeline-side failure; Section 6). We count these two crashed-as-stalled, giving 6/8; excluding the two crashed seeds entirely instead gives 6/6 (Wilson $\approx [0.61, 1.0]$). The cohorts also have unequal n, and the all-seeds mean is dominated by how many seeds escaped: including the two crashed seeds it is 0.63; excluding them it is 0.83 (mean over the 6 surviving seeds, all of which escaped), versus baseline 0.30 ($n = 5$). Finally, UCB-LP was selected for expansion to $n = 8$ *because* it won the initial 3-seed screen (Table 11), so its expanded-cohort mean is an optimistically biased

Table 4: Intervention sweep on `open_large`. We report test beam at the validation-selected checkpoint for each seed (checkpoint chosen by val beam on the 30% split; test beam on the disjoint 70% split). Escape = *beam* ≥ 0.2. Mean is reported both over all seeds (mixing escape and stall) and over escape seeds only, since the bimodal regime makes the all-seeds mean a noisy summary. *Crash convention:* the two UCB-LP seeds that crashed on a SymPy/mpmath overflow were already stalled pre-crash; we count them as stalls (6/8), and parenthetically report the figure excluding them (6/6). The Fisher/Wilson statistics on these counts are in the text. Headline cohorts only; full sweep in Appendix A.12. LP follows Matiisen et al. (2019); oracle = SymPy on the same test set.

Cohort	n	Escape rate	Mean (all)	Mean (escape)	[min, max] escape
Baseline (full stack)	5	3/5	0.30	0.47	$[0.27, 0.83]$
LP curriculum (vanilla)	3	2/3	0.40	0.58	–
UCB-LP curriculum	**8**	**6/8** (6/6)	**0.63**	**0.83**	$[0.69, 0.91]$
Oracle (SymPy)	–	–	1.00	1.00	–

estimate of a selected arm; cross-cohort comparisons here are exploratory, not confirmatory. All numbers are val-best per seed on a 30/70 split (seed 42; Appendix A.13).

Takeaway. The class axis appears to be the relevant axis for intervention: anti-loop penalties and dataset rebalancing leave it unaddressed; vanilla LP modestly shifts the mean (+0.10); UCB-LP raises the escape-seed mean to 0.83 (vs 0.47) and the escape rate from 3/5 to 6/8. This escape-rate change is not statistically significant at this n (Fisher exact $p \approx 1.0$); we read UCB-LP as partially mitigating the bimodality and as a promising direction, not as a demonstrated fix. Figure 2 shows the per-seed distribution: bimodality is preserved (stalls ~ 0.02) but the escape mode is more populated and higher under UCB-LP; Figure 1 (bottom row) overlays the UCB-LP beam curves on baseline, with the wider per-intervention scatter in Appendix A.12.

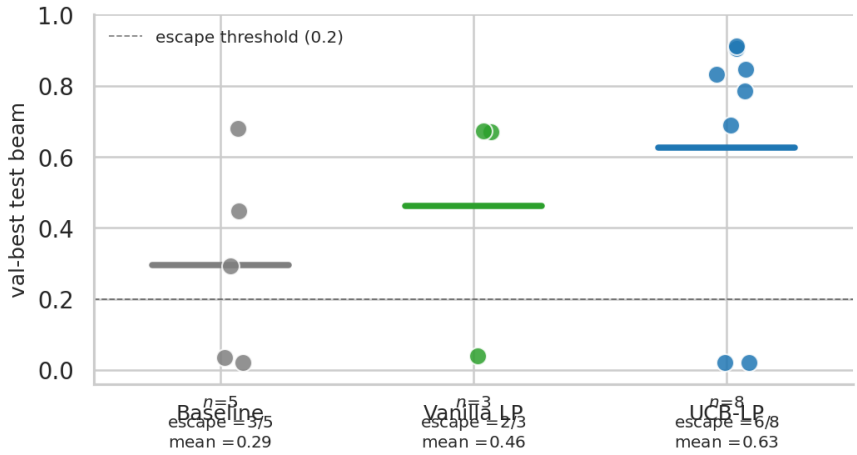

Figure 2: **Per-seed val-best test beam on `open_large`.** Each dot is one training run; horizontal bar is the cohort mean. The escape mode (beam ≥ 0.2; escapes cluster near 0.7–0.9) is more populated and reaches higher values under UCB-LP, while the stall mode at $\sim$0.02 is preserved (UCB-LP: 2/8 stalls, both from crashed seeds whose pre-crash trajectories were already stalled).

4.2.4 Learning the change of variables

π_{cov} is the generator that supplies the substitution when the main policy invokes the CoV macroaction. It is not the main learning challenge of this paper: the four CoV families have known closed-form depressions, and π_{cov} is trained by supervised learning on those targets, so that the RL problem isolates *when* the policy

invokes a CoV and *how* it composes one with ordinary steps – without the substitution generator being the bottleneck.

Task, data, and training. π_{cov} maps an equation to its depression substitution. It is trained, by supervised learning against the known closed-form target, on dedicated datasets that instantiate the depressable forms of the four families with coefficient symbols drawn as permutations from a fixed pool, under random symbol-renaming augmentation. No numeric coefficients appear, so π_{cov} cannot memorize values and must learn the *structural* depression rule. The largest split, `cov_large`, holds 774 training and 192 held-out test equations, the latter symbol-permutations disjoint from training. π_{cov} emits the substitution's expression tree directly, as a sequence of grammar productions in prefix order over $\{+, -, \times, \div, \log\}$, the variable x, the integers $\{2, 3, 4\}$, and a COPY token naming a coefficient symbol (encoding and a worked example in Appendix A.7); it thus chooses the substitution's structure itself rather than filling a fixed template.

Results. On the 192 held-out test equations, π_{cov} produces the correct substitution for **100**%, identically across eight random seeds, under *deterministic* greedy decoding – no search required. We emphasise the scope of this result: π_{cov} *reproduces known closed-form depressions under supervision* on a restricted benchmark, and its held-out set differs from training only by symbol permutation, so this measures generalization to renamed coefficients, not to novel substitution *structures*. Generalization to genuinely new substitution structures (and to symbol pools and template families beyond those seen in training) is untested and is future work. As shown in Section 4.2.2 (config C), π_{cov} is moreover interchangeable with a CAS depression and is not load-bearing for end-task accuracy.

5 Conclusion

We presented the first reinforcement-learning agent to solve *nonlinear* and *open* symbolic equations step-by-step from reward alone, learning *when* to change variables and composing CoV with ordinary algebraic steps under a single policy. It reaches 0.79 beam-decoded test accuracy on open quadratic, cubic, quartic, and exponential families – beyond the linear case prior RL is confined to – and matches the prior best on closed CommonCore equations, while exposing a fully verifiable trace that one-shot neural solvers and a bare CAS do not provide. The principal open problems are the gap between decode-time and training-time search (a natural fit for expert iteration) and the scale-induced seed-level bimodality, which a learning-progress curriculum only partially mitigates. We see the formulation as a step toward RL for the procedural reasoning that differential-equation reductions and prover tactics require.

6 Limitations

Several limitations double as concrete next steps.

The underlying equations are CAS-solvable. Every equation we report is solved in milliseconds by SymPy; our contribution is the RL formulation and the learned step-by-step procedure, not the underlying solvability. The framework's value is in downstream settings – differential-equation solving, theorem-prover tactics, didactic solvers – where the trace matters; on bare equation-solving a CAS comparison is unfavourable and would be misleading.

Residual variance under UCB-LP. UCB-LP partially mitigates the bimodality, driving 6/8 runs into the escape regime (mean *beam*$=0.83$, range 0.69–0.91); the escape-rate change is a trend, not a powered result (Fisher $p \approx 1.0$; following small-sample deep-RL evaluation guidance we report per-seed outcomes rather than relying on point estimates (Henderson et al., 2018; Agarwal et al., 2021)), but two seeds still stall (val-best *beam*≤ 0.05 before a SymPy/mpmath overflow on large rational coefficients ended both at 3.5–3.75M steps; both were already stalled pre-crash). Tighter curricula – e.g. Thompson-sampling class gating Wu et al. (2026) or coupled class/within-class progress – and a hardened numerical pipeline are concrete next steps. The residual 2/8 stall rate and the 0.09 gap to oracle (best 0.91 vs 1.00) are the principal open problems on `open_large`.

Structural ceilings and the weak class. The cubic class was unsolvable until we added a cube-root action, and analogous gaps may exist outside our four CoV templates (e.g. quintics, with no radical solution): the

action set bounds the solvable class in a way learning cannot overcome. Separately, `abel_level3` stays weakest (0.53 full-stack on `open_small`; escape-seed mean 0.39, range $[0.07, 0.55]$, on `open_large`); its dominant failure is the policy-looping pattern of Section 4.2.3, pointing to reward shaping rather than an action-set gap.

Search at evaluation, not training. Beam search dominates final accuracy yet is applied only at decode time – the policy trains for greedy rollouts. Integrating search into training (expert iteration / AlphaZero-style self-improvement Silver et al. (2018)) is the natural and most promising direction beyond this paper.

Broader Impact Statement

This work applies RL to step-by-step symbolic equation solving. The equations are already solvable in milliseconds by computer algebra systems, so the method confers no new equation-solving capability and poses no direct misuse risk; its benefits are indirect, in the verifiable traces it produces for tutoring and for procedural reasoning in ODE solving and prover tactics.

Reproducibility Statement

The MDP formulation, state/action encoding, and reward are specified in Section 3; the TreeMLP architecture, pseudocode, and a minimal implementation are in Appendix A.10. PPO and method hyperparameters are tabulated in Appendix A (Tables 5 and 6), and dataset generation is described in Section 4.1 (closed) and Sections 4.2–4.2.4 (open / π_{cov}), with the rational-equation construction in Appendix A. The validation-based checkpoint-selection protocol and the 30/70 split (seed 42) are given in Appendix A.13; per-seed counts and the crash-counting convention are stated alongside each cohort. Code and datasets will be released upon acceptance.

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2):235–256, 2002.

Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, Daniel Guo, Bilal Piot, Steven Kapturowski, Olivier Tieleman, Martín Arjovsky, Alexandre Pritzel, Andew Bolt, and Charles Blundell. Never give up: Learning directed exploration strategies. In *International Conference on Learning Representations (ICLR)*, 2020.

Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. Exploration by random network distillation. In *International Conference on Learning Representations (ICLR)*, 2019.

Lennart Dabelow and Masahito Ueda. Symbolic equation solving via reinforcement learning. *Neurocomputing*, 613:128732, 2024.

Yihong Dong, Xue Jiang, Yongheng Tao, et al. RL-PLUS: Countering capability boundary collapse of LLMs in reinforcement learning with hybrid-policy optimization. *arXiv preprint arXiv:2508.00222*, 2025.

Kevin Ellis, Catherine Wong, Maxwell Nye, Mathias Sablé-Meyer, Luc Morales, Luke Hewitt, Luke Cary, Armando Solar-Lezama, and Joshua B. Tenenbaum. DreamCoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 2021.

Zhengyao Gu, Jonathan Light, Raul Astudillo, Ziyu Ye, Langzhou He, Henry Peng Zou, Wei Cheng, Santiago Paternain, Philip S. Yu, and Yisong Yue. Actor-curator: Co-adaptive curriculum learning via policy-improvement bandits for RL post-training. *arXiv preprint arXiv:2602.20532*, 2026.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *AAAI Conference on Artificial Intelligence*, 2018.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017.

Guillaume Lample and François Charton. Deep learning for symbolic mathematics. In *International Conference on Learning Representations (ICLR)*, 2020.

Zhening Li, Gabriel Poesia, Omar Costilla-Reyes, Noah Goodman, and Armando Solar-Lezama. LEMMA: Bootstrapping high-level mathematical reasoning with learned symbolic abstractions. In *NeurIPS Workshop on MATH-AI*, 2022.

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher–student curriculum learning. In *IEEE Transactions on Neural Networks and Learning Systems*, 2019.

Aaron Meurer, Christopher P. Smith, Mateusz Paprocki, Ondřej Čertík, Sergey B. Kirpichev, Matthew Rocklin, Amit Kumar, Sergiu Ivanov, Jason K. Moore, Sartaj Singh, Thilina Rathnayake, et al. SymPy: symbolic computing in Python. *PeerJ Computer Science*, 3:e103, 2017.

Kevin P. O'Keeffe. Curiosity-driven reinforcement learning for symbolic equation solving. *NeurIPS 2025 MATH-AI Workshop; arXiv:2510.17022*, 2025.

Deepak Pathak, Pulkit Agrawal, Alexei A. Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, 2017.

Brenden K. Petersen, Mikel Landajuela, T. Nathan Mundhenk, Claudio P. Santiago, Soo K. Kim, and Joanne T. Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations (ICLR)*, 2021.

Gabriel Poesia and Noah D. Goodman. Peano: Learning formal mathematical reasoning. In *Philosophical Transactions of the Royal Society A*, volume 381, 2023.

Gabriel Poesia, WenXin Dong, and Noah Goodman. Contrastive reinforcement learning of symbolic reasoning domains. In *Advances in Neural Information Processing Systems*, volume 34, pp. 17729–17741, 2021.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2023.

Miguel Suau, Matthijs T. J. Spaan, and Frans A. Oliehoek. Bad habits: Policy confounding and out-of-trajectory generalization in RL. *Reinforcement Learning Journal*, 4:1711–1732, 2024.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

Yuning Wu, Ke Wang, Devin Chen, and Kai Wei. Hindsight-anchored policy optimization: Turning failure into feedback in sparse reward settings. *arXiv preprint arXiv:2603.11321*, 2026.

Rosie Zhao, Tian Qin, David Alvarez-Melis, Sham Kakade, and Naomi Saphra. Random scaling of emergent capabilities. *arXiv preprint arXiv:2502.17356*, 2025.

A Supplementary material

A.1 Macroactions

Here we justify why the three macroactions $(\text{expand}, \text{None})$, $(\text{collect}, x)$, $(\text{mul}, -1)$ discussed in the main text are *required* to solve the equations we consider, not mere conveniences for the RL agent. Consider the action set *without* these macroactions:

$$
\begin{aligned}
O &= \{\text{add},\ \text{sub},\ \text{mul},\ \text{div}\} \\
&\quad \cup \{\text{square},\ \text{sqrt},\ \text{exp},\ \text{log},\ \text{sin},\ \text{cos},\ \text{asin},\ \text{acos}\}, && (13)\\
T &= \text{SubExpr}(\text{lhs}) \cup \text{SubExpr}(\text{rhs}), && (14)\\
A &= (O \times T). && (15)
\end{aligned}
$$

Now consider the equation $dx + c(ax + b) = e$. The agent must distribute c into the bracketed term $ax + b$, which the action set above cannot do: expand is required. Once expanded to $dx + cax + cb = e$, one needs to factor the x common to the first two terms—this is what $(\text{collect}, x)$ provides. Finally, consider $-x = b$: since -1 is not in our term set, $(\text{mul}, -1)$ is needed to isolate x. (Alternatively, -1 could be added to the term set, but we chose to keep the term set purely symbolic.)

A.2 Dataset generation: rational equations

To supplement the recursive datasets, we constructed rational equations of the form

$$\frac{ax + b}{cx + a} + b = 0,$$

where a, b, c are (non-zero) symbolic coefficients. We excluded degenerate cases such as denominators that vanish identically or cancel trivially with numerators.

Examples of retained equations include:

$$\frac{x + b}{cx + a} = 0 \quad \Rightarrow \quad x = -b,$$

$$\frac{ax + b}{cx + a} + b = 0 \quad \Rightarrow \quad x = \frac{-b - ab}{a + cb},$$

$$\frac{ax + b}{cx + a} - c = 0 \quad \Rightarrow \quad x = \frac{-b + ac}{a - c^2}.$$

These rational forms were merged into both the small and large datasets, with their proportion limited to preserve diversity across other functional forms (logarithmic, trigonometric, exponential, etc.).

A.3 Illegal actions

The main issue was to avoid illegal division by zero. This is not as simple as precluding $(\text{div}, 0)$ from the action set. We also had to prohibit *hidden* division by zero—for example, dividing by $(x+a)$ or $(x+b)$ when the equation has the form $(x + a)(x + b) = 0$, since one of the factors will vanish at the solution. To handle this, we check whether the current equation factors as $P(x)Q(x) \cdots = 0$ with $P, Q, \ldots$ polynomial in x, and remove $(\text{div}, P), (\text{div}, Q), \ldots$ from the action set for that state.

A.4 Hyperparameters and compute

We used Stable Baselines 3's default hyperparameters for PPO (Table 5), together with the architecture and method constants stated in the body and consolidated in Table 6.

Compute budget. All experiments run on a single 24-core CPU machine (no GPU). The PPO `device` setting is left at its Stable-Baselines3 default of `'auto'` (Table 5), but because the machine has no GPU this resolves to CPU for every run reported here. Each `open_large` training is 5×10^6 steps (`open_small` uses 3×10^6) with $n_{\text{envs}} = 1$, taking 20–30 wall-hours per seed (one CPU core per training; helpers for in-training eval and post-hoc beam-curve eval add up to four more cores transiently). The full $n = 8$ UCB-LP cohort plus $n = 5$ baseline cohort plus the LP-only and intervention-sweep cohorts amount to roughly 1,500–2,000 core-hours of training, dominated by the 5×10^6-step open_large runs. Post-hoc beam-curve evaluations for Figure 1 add an additional ~ 50 core-hours. Closed-equation experiments (10^7 steps each) account for another ~ 300 core-hours across all cohorts. Total: $\sim 2{,}500$ core-hours, no GPU.

Algorithm	**Hyperparameters**
PPO	*learning_rate* = 0.0003, *n_steps* = 2048, *batch_size* = 64, *n_epochs* = 10, *gamma* = 0.99, *gae_lambda* = 0.95, *clip_range* = 0.2, *ent_coef* = 0.0, *vf_coef* = 0.5, *max_grad_norm* = 0.5, *normalize_advantage* = *True*, *device* $=$' *auto*'
PPO-tree	*learning_rate* = 0.0007, *n_steps* = 5, *gamma* = 0.99

Table 5: Default PPO hyperparameters from Stable Baselines 3. The `device='auto'` default resolves to CPU on our no-GPU machine.

Constant	**Value**
TreeMLP message-passing rounds K	3
TreeMLP hidden size H	128
TreeMLP embedding dim	64
TreeMLP pooling	mean or max (Sec. 3.1)
max state length L	50
action-set cap $\lvert A\rvert$	50
action-diversity penalty α	0.1
value-beam (λ, β)	$(1, 0)$
beam width	5
buffer-refresh period	every 20 rollouts (discard oldest half)
CoV per-episode cap	3
UCB-LP exploration coeff. β	2.0

Table 6: Architecture and method constants, consolidated from the body and the training configuration.

A.5 Closed-equation analysis

This appendix expands the closed-equation results of Section 4.1: a per-structural-type accuracy breakdown, the dominant failure modes, and how performance changes from the small to the large dataset.

Per-type accuracy. The "small" dataset contains a heterogeneous mix of equation types. Bucketing the test set by structural class and re-evaluating the trained PPO-TREE-RC-BUF agent reveals a clear pattern (Table 7): linear and radical equations are nearly fully solved (≥ 0.88), polynomial equations of degree 2–4 sit in the 0.75–0.82 range, while transcendental forms—particularly trigonometric and logarithmic—are the hardest. The trigonometric bucket (113 equations, 72% accuracy) accounts for most of the remaining failures. Note that the polynomial buckets contain only "depressed" forms (e.g. $ax^2 + c$, $x^4 + c$) that arise naturally from recursive action application; *true* open polynomials with a linear cross-term ($ax^2 + bx + c$) are not in this dataset and are instead studied in Section 4.2.

Failure modes. Manual inspection of the failed cases (75 of 387) shows three recurring patterns: (i) trig equations that require an arccos/arcsin step the agent never selects, (ii) log equations whose simplification creates a transcendental that breaks the closed-action set, and (iii) deep polynomials where the agent exceeds the per-episode action budget despite being on a productive path.

Table 7: Greedy test accuracy of PPO-TREE-RC-BUF on the small dataset, broken down by equation class. "poly-deg$k$" includes only the depressed forms reachable by the closed-equation action set (e.g. $ax^2 + c$, $x^4 + c$); equations with a linear cross-term are out of this dataset.

Type	n	solved	acc
radical (e.g. $\sqrt{x}$)	47	43	0.915
linear (poly-deg1)	110	97	0.882
quadratic (poly-deg2)	34	28	0.824
log	77	58	0.753
poly-deg4	4	3	0.750
trig	113	81	0.717
overall	387	312	0.806

Scaling to the large dataset. The same per-type evaluation on the "large" dataset (1000 test equations of depth < 5) reveals a uniform drop in every category: linear $0.88 \to 0.60$, quadratic $0.82 \to 0.40$, log $0.75 \to 0.27$, trig $0.72 \to 0.36$, radical $0.92 \to 0.52$. The drop is largest for deep polynomials (poly-deg4: $0.75 \to 0.08$). Going from depth < 4 to depth < 5 approximately quadruples the test-set size and adds substantially deeper nesting, so the harder cases are genuinely harder, not just more numerous.

Generalization to CommonCore. On the CommonCore (Poesia) benchmark, PPO-TREE-RC-BUF reaches 0.93 greedy test accuracy on the full benchmark (Table 1), matching ConPoLe's 0.925. The non-buffer PPO-TREE-RC variant scores 0.85 on the full benchmark; restricting the denominator to non-degenerate templates – excluding edge-case equations that lack a free x in the polynomial decomposition, which the closed-action set cannot solve – raises its greedy accuracy to 0.955 (linear 0.98, small rational subset 1.00), and ≥ 0.98 on the solvable subset. The 0.85 and 0.955 figures for PPO-TREE-RC thus differ only in the denominator (full benchmark vs. non-degenerate subset); both use greedy decoding.

A.6 Open-equation embeddings (converged vs. stalled seeds)

To diagnose whether the bimodal failure on `open_large` (Section 4.2.3) is a representation problem or a policy problem, we extract the pooled TreeMLP graph embedding for each test equation and project it to two dimensions with PCA and t-SNE. Both the converged seed (`seed109000`; $test_{\text{beam}} = 0.81$) and a stalled seed (`seed108000`; $test_{\text{beam}} = 0.03$) produce clean class clusters (quadratic / cubic / quartic / exponential each form a tight clump in t-SNE space; `abel_level3` is more diffuse). The encoder thus learns the equation-family structure in both cases; the failure is in the policy/value head's ability to act on it, not in the input representation.

A.7 Change-of-variables grammar (π_{cov})

π_{cov} decodes the substitution as a prefix sequence of grammar productions; production arities make decoding self-delimiting (no explicit length field or brackets needed). For example, $x - b/(2a)$ is emitted as the token sequence SUB X DIV COPY MUL INT2 COPY: SUB takes two children (x and the quotient), DIV takes the copied symbol (b) and the product $2a$, MUL takes INT2 and a second copied symbol (a), and each COPY names a coefficient symbol of the current equation. Because the model chooses each production – whether to subtract, what to divide by, which coefficients to use – it selects the substitution's structure rather than filling a fixed template.

A.8 Open-equation solution traces

Below are full greedy solution traces for one held-out test equation from each source class of `open_large`, produced by the converged `seed109` agent. The four change-of-variables classes (quadratic, cubic, quartic, exponential) share the same recipe up to constants (COV $\to$ RELABEL $\to$ mul(-1) $\to$ mul x $\to$ collect x $\to$ div $\to$ RELABEL); only the exponential class requires a *second* CoV at step 4. The depth column counts active CoV substitutions.

step	action	depth	current lhs (rhs = 0)
quadratic – $ax^2 + 2cx + 5d = 0$			
1	COV	1	$ax^2 + 5d - c^2/a$
2	RELABEL	1	$a + bx^2$
3	mul(-1)	1	$-a - bx^2$
4	mul x	1	$x(-a - bx^2)$
5	collect x	1	$x(-a - bx^2)$
6	div($-a - bx^2$)	1	x
7	RELABEL	1	x [$= -c/a$]
cubic and **quartic**: identical 7-step recipe (only the degree of x differs); see text.			
exponential (nested CoV) – $2b\,e^x + 2d - 6f\,e^{-x} = 0$			
1	COV	1	$2b/x + 2d - 6fx$
2	mul x	1	$x(2b/x + 2d - 6fx)$
3	EXPAND	1	$2b + 2dx - 6fx^2$
4	COV (nested)	2	$2b + d^2/(6f) - 6fx^2$
5	RELABEL	2	$a + bx^2$
6	mul(-1)	2	$-a - bx^2$
7	mul x	2	$x(-a - bx^2)$
8	collect x	2	$x(-a - bx^2)$
9	div($-a - bx^2$)	2	x
10	RELABEL	2	x [$= -\log(d/f) + \log 6$]
abel_level3 (closed; no CoV needed) – $ab^2x^2 = 0$			
1	RELABEL	0	ax^2
2	div a	0	x^2
3	div x	0	x [$= 0$]

Table 8: Per-class greedy solution traces from the converged `open_large` agent (`seed109`, full method stack). The four change-of-variables classes share the same 7-step recipe up to constants; the exponential class is the only one that recruits a *nested* CoV (step 4) after the inner equation becomes a quadratic. The closed-form `abel_level3` example needs no CoV at all and solves in 3 steps.

A.9 Closed-equation solution traces

Below are the solution traces for each equation in the fixed *closed*-equation environment of Section 4.1. Notice the solution trace is not always optimal. For instance, $ax + b = 0$ could be solved with $(sub, b), (div, a)$, but instead the agent selects (sub, ax) and then $(mul, -1)$. "Truediv" is the SymPy notation for divide.

1. Equation: $a + x = 0$

```
Step 1: a + x = 0 | subtract, a
Solved: x = -a
```

2. Equation: $ax = 0$

```
Step 1: a*x = 0 | div, a
Solved: x = 0
```

3. Equation: $ax + b = 0$

```
Step 1: ax + b = 0 | subtract, ax
Step 2: -ax = b | multiply, -1
Step 3: ax = -b | div, a
Solved: x = -b/a
```

4. Equation: $a/x + b = 0$

```
Step 1: a/x + b = 0 | subtract, b
Step 2: a/x = -b | truediv, 1/x
Step 3: -b*x = a | truediv, b
Step 4: -x = a/b | multiply, -1
Solved: x = -a/b
```

5. Equation: $c(ax + b) + d = 0$

```
Step 1: c*(ax + b) + d = 0 | expand, None
Step 2: acx + bc + d = 0 | subtract, acx
Step 3: -acx = bc + d | multiply, -1
Step 4: acx = -bc - d | truediv, c
Step 5: ax = (-bc - d)/c | truediv, a
Solved: x = (-bc - d)/(ac)
```

6. Equation: $c + d/(ax + b) = 0$

```
Step 1: c + d/(ax + b) = 0 | subtract, c
Step 2: d/(ax + b) = -c | multiply, (ax + b)
Step 3: d = -c (ax + b) | expand, None
Step 4: d = -c ax - c b | multiply, -1
Step 5: -d = c ax + c b | subtract, c b
Step 6: -d - c b = c ax | truediv, c
Step 7: (-d - c b)/c = ax | truediv, a
Step 8: ((-d - c b)/c)/a = x | truediv, 1/a
Solved: x = (-d - c b)/(c a)
```

7. Equation: $cx + d + e(ax + b) = 0$

```
Step 1: cx + d + e(ax + b) = 0 | expand, None
Step 2: aex + be + cx + d = 0 | collect, x
Step 3: be + d + x*(ae + c) = 0 | subtract, x(ae + c)
Step 4: -x(ae + c) = be + d | truediv, ae + c
Step 5: -x = (be + d)/(ae + c) | multiply, -1
Solved: x = -(be + d)/(a*e + c)
```

8. Equation: $e + (ax + b)/(cx + d) = 0$

```
Step 1: e + (ax + b)/(cx + d) = 0 | truediv, 1/(cx + d)
Step 2: (e + (ax + b)/(cx + d))(cx + d) = 0 | expand, None
Step 3: ax + b + cex + de = 0 | collect, x
Step 4: b + de + x*(a + ce) = 0 | subtract, x(a + ce)
Step 5: -x(a + ce) = b + de | expand, None
Step 6: -ax - cex = b + de | collect, x
Step 7: x*(-a - ce) = b + de | truediv, -a - ce
Solved: x = (b + de)/(-a - c*e)
```

A.10 TreeMLP Details

Architecture. TreeMLP embeds node IDs, projects to a hidden size, and performs K message-passing stages over the (directed) expression tree. At each stage it sums incoming neighbor states, concatenates the result with the fixed input embedding, and applies a two-layer MLP with ReLU. A masked global pool (mean or max) produces the graph embedding exposed to the policy/value heads.

Algorithm 1: TreeMLP message passing and pooling

Require: node IDs $\mathbf{X}$, edges $(\mathrm{src}, \mathrm{dst})$, masks $m_{\mathrm{node}}, m_{\mathrm{edge}}$, steps K

$\mathrm{emb} \leftarrow \mathrm{Proj}(\mathrm{Embed}(\mathbf{X}))$ ▷ shape $[B, N, H]$
$h \leftarrow \mathrm{emb}$
for $k = 1$ **to** K **do**
  $\mathcal{E} \leftarrow \{(i \to j) \mid m_{\mathrm{edge}}(i \to j) = 1\}$
  $\mathrm{agg}[j] \leftarrow \sum_{(i \to j) \in \mathcal{E}} h[i]$ ▷ scatter-sum
  $h \leftarrow \mathrm{MLP}\big([\mathrm{emb}, \mathrm{agg}]\big)$
  $h \leftarrow h \odot m_{\mathrm{node}}$ ▷ mask padded nodes
end for
return $\mathrm{pool}(h, m_{\mathrm{node}})$ ▷ mean or max

Listing 1: TreeMLP forward pass (minimal).

```
def forward(self, obs: dict) -> torch.Tensor:
    node_ids  = self._to_device(obs["node_features"], self.device)
    edge_index= self._to_device(obs["edge_index"], self.device)
    node_mask = self._to_device(obs["node_mask"], self.device)
    edge_mask = self._to_device(obs["edge_mask"], self.device)

    node_idx = self._id_to_idx(node_ids) if node_ids.dtype != torch.long else node_ids
    B, N = node_idx.shape
    _, _, E = edge_index.shape

    emb = self.embed(node_idx)                    # [B,N,embed_dim]
    emb = self.embed_proj(emb)                    # [B,N,H]
    h = emb.clone()

    src = edge_index[:, 0, :].long()             # [B,E]
    dst = edge_index[:, 1, :].long()             # [B,E]
    batch_idx = torch.arange(B, device=self.device).unsqueeze(1).expand(B, E)

    for _ in range(self.K):
        src_flat = src.reshape(-1)
        dst_flat = dst.reshape(-1)
        b_flat   = batch_idx.reshape(-1)
        m_flat   = edge_mask.reshape(-1).bool()

        src_flat, dst_flat, b_flat = src_flat[m_flat], dst_flat[m_flat], b_flat[m_flat]
        src_idx = b_flat * N + src_flat
        dst_idx = b_flat * N + dst_flat

        h_flat   = h.reshape(B * N, self.hidden_dim)
        messages = h_flat[src_idx]                          # [M,H]

        agg_flat = torch.zeros_like(h_flat)                 # [B*N,H]
        agg_flat.index_add_(0, dst_idx, messages)           # sum incoming
        agg = agg_flat.reshape(B, N, self.hidden_dim)       # [B,N,H]

        h = self.node_mlp(torch.cat([emb, agg], dim=-1))    # [B,N,H]
        h = h * node_mask.unsqueeze(-1).float()

    if self.pooling == "mean":
        denom = node_mask.sum(dim=1, keepdim=True).clamp(min=1).float()
        g = h.sum(dim=1) / denom
```

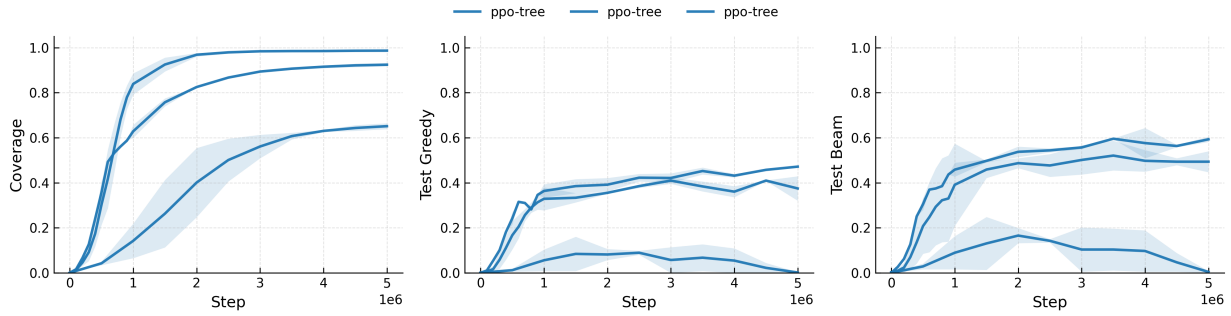

Figure 3: TreeMLP outperforms GCN and GraphSAGE on the small datasets. Mean of three random seeds with shading to min/max.

```
    else:
        if self._minus_inf is None or self._dtype_cache != h.dtype:
            self._minus_inf = torch.finfo(h.dtype).min
            self._dtype_cache = h.dtype
        g = h.masked_fill(~node_mask.unsqueeze(-1), self._minus_inf).max(dim=1)
            .values

    return self.proj(g)
```

Implementation notes. We use fixed-size padding with boolean masks for nodes/edges, scatter-add for neighbor aggregation, and pool with mask awareness. All tensors are moved to CUDA when available; no MPS is used.

A.11 Ablations

We summarize the ablation findings; the corresponding learning curves are in Figures 4 and 3.

- Replacing dense complexity-delta rewards with sparse outcome-only rewards (and disabling the inverse-frequency curriculum) substantially degrades performance. The main drawback of sparse rewards is *far* longer run times: we had to introduce a 1-second-per-step wall-clock timeout to keep training tractable.
- TreeMLP outperforms a vanilla GCN and GraphSAGE on the small dataset in single-seed runs (Figure 3); the tree-aware aggregation appears helpful here, though this comparison is suggestive, not conclusive.
- Curiosity bonuses (ICM, RND, NGU) do not improve TreeMLP learning on `abel_level3` (Figure 5); the body discusses this negative result in Section 3.2. The bonus runs are single-seed and so are suggestive, not conclusive.

A.12 Deferred main-text figures and tables

This section collects figures and tables deferred from the main text. Each is referenced inline from the appropriate main-text passage.

Expression trees (referenced from Sec. 4 state encoding).

Curiosity-bonus negative result (referenced from §4.1). See Figure 5 and the ablation bullet above; the negative result and its plausible explanation are discussed in Section 3.2.

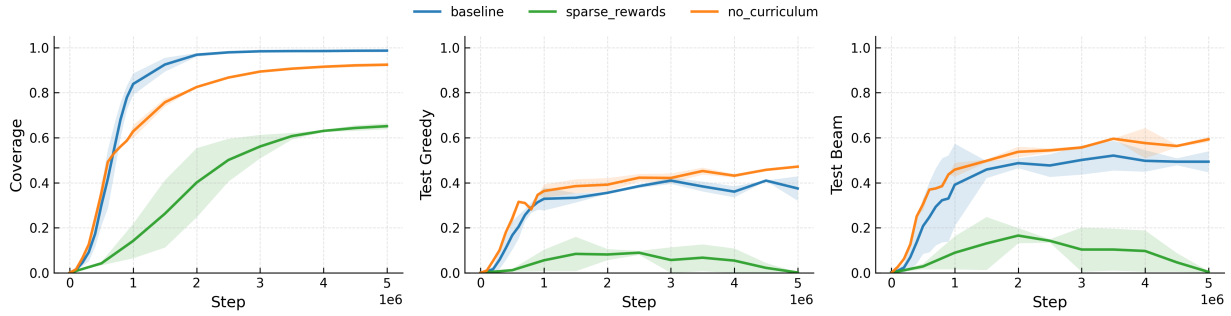

Figure 4: Inductive-bias ablations on the small closed-equation dataset. Each curve removes one component (dense complexity-delta reward + inverse-frequency curriculum, relabel-constants, or the success-replay buffer) from the full PPO-TREE-RC-BUF stack; y-axis is greedy test accuracy over training steps (mean of 3 seeds, min/max shading). Conclusion: removing the dense reward/curriculum or the replay buffer substantially degrades learning, so each bias is load-bearing on this dataset.

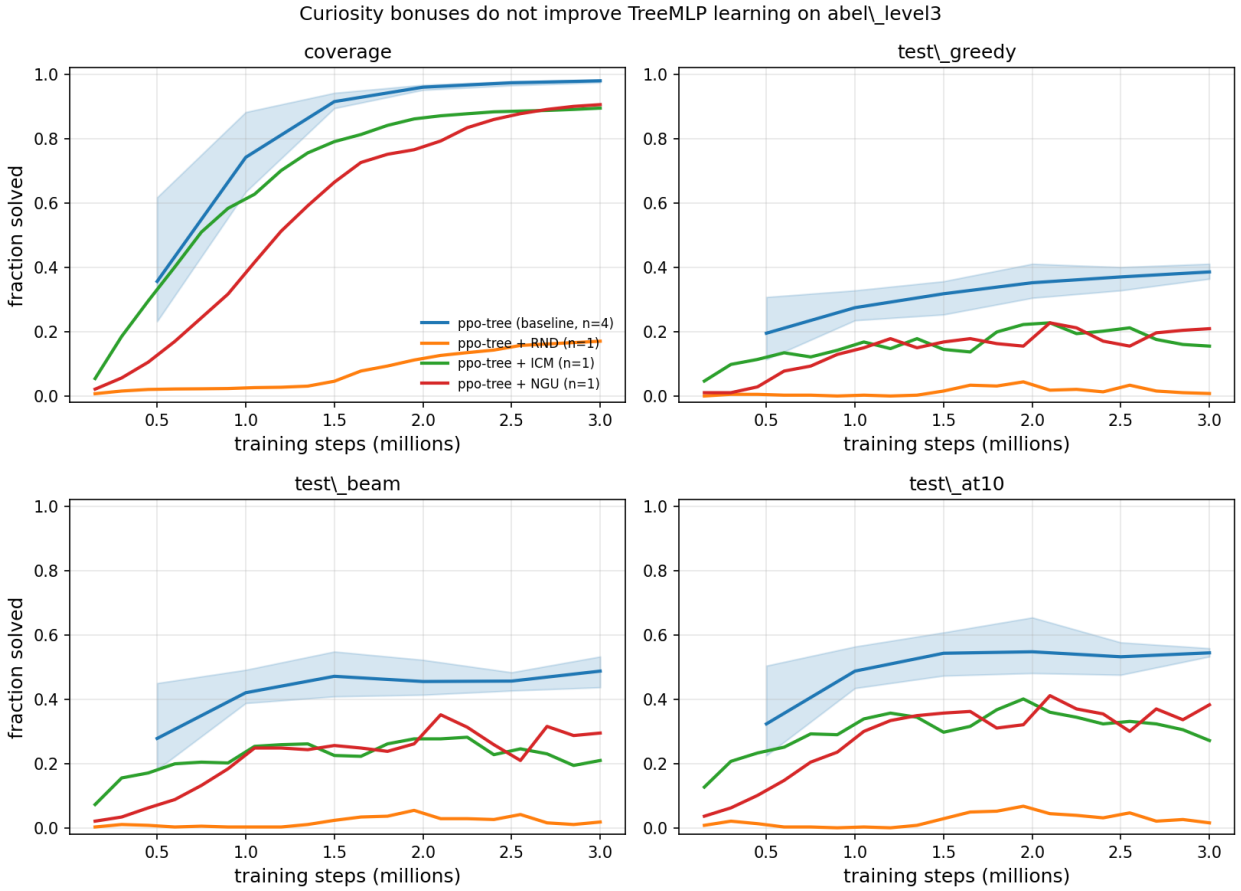

Figure 5: Curiosity bonuses (ICM, NGU, RND) do not improve TreeMLP learning on `abel_level3` (single-seed bonus runs vs. an n=4-seed no-curiosity baseline; y-axis test metrics over training steps). No bonus matches the baseline; RND fails to learn. These single-seed results are suggestive, not conclusive.

TreeMLP embeddings cluster by family (referenced from §4.1).

Per-class accuracy on `open_small` (referenced from Section 4.2.2).

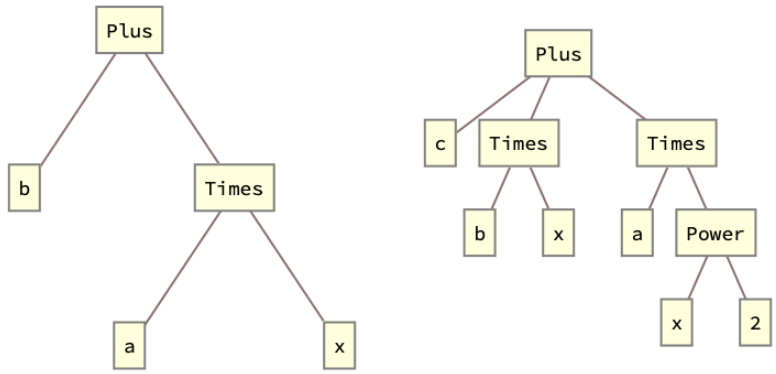

Figure 6: Expression trees for $ax + b$ and $ax^2 + bx + c$. The agent's state vector is a preorder traversal of these trees, padded to length $L = 50$.

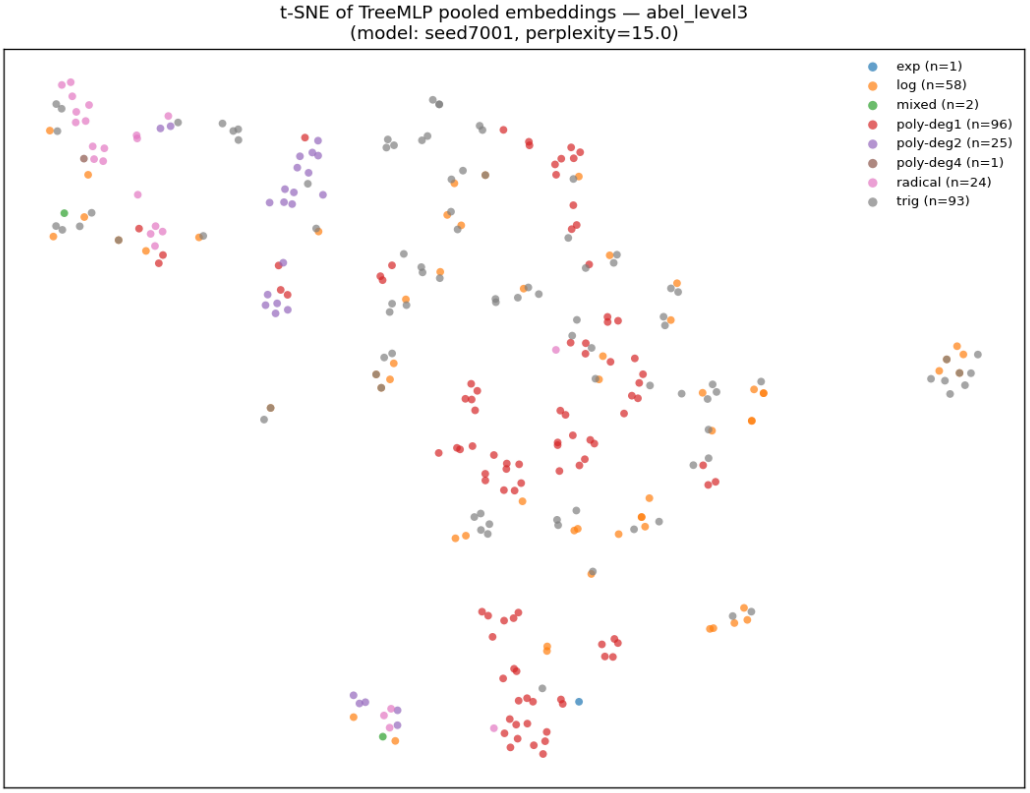

Figure 7: t-SNE projection of TreeMLP pooled embeddings for 300 small-dataset equations, colored by structural type. Clusters correspond to equation family rather than to specific coefficient assignments.

Per-class accuracy on `open_large` across escape seeds (referenced from Section 4.2.2).

Bimodality per-seed dots (referenced from Section 4.2.3).

Full 9-row intervention sweep (referenced from Section 4.2.3).

Table 9: Per-class test accuracy on `open_small` (value-guided beam). Baseline is the single-seed PPO-TREE-RC-BUF-COV reference ($n = 1$); full-stack is the 4-seed mean of seeds completing the full budget (the 5th early-stopped seed is excluded here, whereas the headline 0.79 in Section 4.2.2 is the 5-seed mean that includes it – the two are therefore not directly comparable). The n-weighted overall of these 4-seed per-class values is ≈ 0.85 on the full 91 equations; the headline 0.79 is on the 64-equation test slice.

class (n)	baseline	full stack
quadratic (15)	0.00	0.97
cubic (15)	0.00	1.00
quartic (15)	0.00	1.00
exponential (15)	0.00	0.88
abel_level1 (1)	1.00	1.00
abel_level2 (10)	0.50	0.75
abel_level3 (20)	0.25	0.53

Table 10: Per-class `open_large` test accuracy (value-guided beam, 200 eqns/class) across the 3 escape seeds (*overall* $\in [0.79, 0.91]$); the 6 stall seeds are ≈ 0 on every class. The four CoV classes saturate while `abel_level3` remains weak and high-variance – the open_small pattern, sharpened at scale.

class ($n = 200$)	escape-seed mean [min, max]
quadratic	0.98 [0.98, 0.99]
cubic	0.99 [0.98, 1.00]
quartic	0.99 [0.98, 1.00]
exponential	0.98 [0.96, 1.00]
abel_level3	0.39 [0.07, 0.55]

Three competing hypotheses ruled out (referenced from Section 4.2.3). We rejected the three most natural alternative explanations for the bimodal failure:

- *Capacity loss / dormant neurons* Sokar et al. (2023): at the standard ReDo activation threshold (0.1× per-layer max), *both* stuck and escape seeds have 0% dormant neurons; not a capacity-loss phenomenon.
- *Data-distribution skew*: an alternative training distribution skewed toward `abel_level3` (the "abel-boost" set) reaches *beam* = 0.93 on its matched test split but only 0.04 on the standard test set – the policy overfits to whatever distribution it sees, ruling out a fixable training-data mismatch.
- *Encoder representation collapse*: t-SNE projections of the TreeMLP encoder show clean per-class clusters on *stalled* seeds too (Appendix A.6); the representations are intact.

The bimodality is therefore a *policy-attractor* phenomenon, not a capacity, data, or representation failure.

A.13 Validation-based checkpoint selection protocol

(Referenced from Section 4.2.2.) For every reported standalone-eval beam value, we use a 30/70 validation/test split of the test set with seed 42, evaluate the saved checkpoints on the validation split, select the checkpoint with the highest val beam, and report its test-split beam.

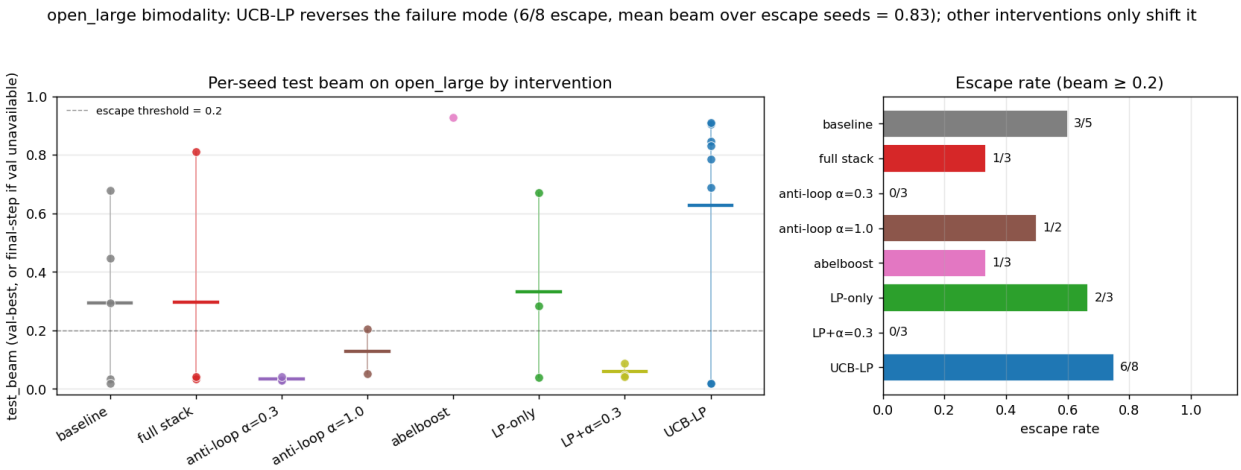

Figure 8: Per-seed beam test accuracy and cohort escape rates from the *initial* 3-*seed screen* on `open_large`. Each dot is a single seed at its validation-best checkpoint; bars show fraction of seeds with *beam*$\geq$0.2 (escape threshold). UCB-LP was the only cohort with 3/3 escape in this screen, which motivated expanding it to $n = 8$ (Table 4, where it reaches 6/8). Numbers in Table 4 supersede this screen for the three headline cohorts.

Table 11: **Initial 3-seed screen** of the full intervention sweep on `open_large` (mean validation-best beam over 3 seeds; escape = *beam* $\geq$ 0.2). All cohorts were first screened at 3 seeds to select interventions; the three headline cohorts were subsequently expanded to $n = 5$ (baseline), 3 (vanilla LP), and 8 (UCB-LP) seeds, with final numbers in Table 4, which *supersede* the corresponding rows here. The non-headline rows (anti-loop, dataset rebalance, LP+anti-loop, LP++) were screened only and not expanded.

Cohort	Mean beam	Escape rate
Baseline (full stack)*	0.34	2/3
Anti-loop (α=0.3)	0.05	0/3
Anti-loop (α=1.0)	0.11	1/3†
Dataset rebalance (abelboost)	0.04	0/3
LP curriculum (vanilla)* Matiisen et al. (2019)	0.40	2/3
LP + anti-loop (α=0.3)	0.07	0/3
LP++ (optim. init + EMA + boot)	0.10	0/3
UCB-LP curriculum*	**0.81**	**3/3**
Oracle (SymPy)	1.00	–

* headline cohort; expanded beyond this 3-seed screen, see Table 4 for the superseding numbers. † one of three seeds crashed mid-run due to a SymPy/mpmath `OverflowError` on a large rational; we report rate over surviving seeds.