FactorLibrary: From Polynomials to Circuits via Recursive Subgoals

Rohan Pandey [*1] **Michael Ruofan Zeng** [*1] **Weikun K. Zhang** [*1] **Kaijie Jin** [1] **Naomi Morato** [1]
Archit Ganapule [1] **Bhaumik Mehta** [1] **Jarod Alper** [1]

Abstract

Finding minimal arithmetic circuits for polynomials over finite fields is a combinatorially hard problem central to algebraic complexity theory. We formulate it as a reinforcement learning problem in two directions, bottom-up and top-down. To address the challenge of a fast-growing combinatorial search space, we introduce `FactorLibrary`, which stores factorizable subexpressions that serve as reusable subgoals across training episodes. We trained a bottom-up agent with Gumbel-PPO-MCTS and two top-down agents with PPO+MCTS and SAC. The PPO+MCTS top-down agent exhibited the most stable performance, finding certified optimal circuits up to complexity 8 with a success rate of 91.8%.

1. Introduction

An *arithmetic circuit* is a sequence of addition and multiplication gates that computes a given polynomial from initial variable nodes (Bürgisser et al., 1997; Shpilka & Yehudayoff, 2010). A circuit is *efficient* if it computes the polynomial with the minimal number of gates. The problem of finding efficient arithmetic circuits is central in algebraic complexity theory, and it also provides a useful testbed for reinforcement learning methods that search over a large combinatorial action space (Valiant, 1979b;a; Lidl & Niederreiter, 1997). A much more difficult search problem of this sort is automatic generation of mathematical proofs, where learned policies must compose many verified symbolic steps (Hubert et al., 2025).

In recent related work, Zhang et al. (2026) formulated the arithmetic circuit problem as a bottom-up reinforcement-learning game. Their "`CircuitBuilder`" agent begins with an empty circuit consisting of only variable and constant nodes, applies one gate at a time, and succeeds when a computed node equals the target polynomial. Zhang et al. compared PPO+MCTS with discrete SAC training architectures and observed that off-policy learning was competitive on easier two-variable tasks, while on-policy search performed better as the number of variables increased.

In this paper, we adopt the *bottom-up* approach of Zhang et al. (2026) and train with Gumbel-PPO-MCTS, where Gumbel search produces improved targets for PPO updates (Danihelka et al., 2022; Schulman et al., 2017). We also study an opposite *top-down* formulation that reverses the direction of search. The agent starts from the target polynomial f and repeatedly proposes addition split points $f = g + h$, relying on the game environment to produce lists of factors of g and h. The agent then continues to recursively propose split points for smaller expressions until the search reaches the variable and constant nodes. The top-down formulation dramatically reduces the effective action space by exploiting deterministic factorization algorithms, and the results demonstrate this reduction clearly.

Both formulations use a `FactorLibrary`, a collection of polynomials that serve as subgoals. It serves as both the agent's short-term memory and a list of recursive subgoals, under the heuristic that if the agent has learned to produce simpler subexpressions, then it should assemble them into the bigger expression. The technique of breaking down goals into recursive subgoals has been considered in Kulkarni et al. (2016), Andrychowicz et al. (2017), and Florensa et al. (2018).

We trained a bottom-up agent with Proximal Policy Optimization (PPO) integrated with Monte Carlo Tree Search (MCTS) and Gumbel, and two top-down agents with PPO+MCTS and Soft Actor-Critic (SAC). We evaluate all three agents on two-variable polynomials over $\mathbb{F}_5$. The bottom-up agent achieves near-perfect success on small circuits (99.2% aggregate up to complexity C_3), but performance degrades sharply, falling to 7.8% at C_{10} due to the combinatorial explosion of the action space. The top-down agents perform significantly better. PPO+MCTS reaches a 91.8% success rate in finding optimal circuits across complexities C_2–C_{10}. On the other hand, the best SAC check-

[*]Equal contribution [1]University of Washington, Seattle, WA, USA. Correspondence to: Michael Ruofan Zeng <zengrf@uw.edu>.

The 3rd *AI for Math Workshop* at the 43rd International Conference on Machine Learning (ICML), Seoul, South Korea, 2026.
Copyright 2026 by the author(s).

point achieved a success rate of 92.8% at roughly one-fifth the compute cost. Both methods far exceed the 40% uniform random baseline, demonstrating that the learned split and factor policies generalize to unseen targets.

We end the introduction by remarking that training RL agents to solve the arithmetic circuit problem with a high success rate is desirable for the following reasons. First, finding efficient circuits for important families of polynomials, such as permanents, is of independent mathematical interest and lies at the heart of algebraic complexity theory (Valiant, 1979a; Bürgisser et al., 1997; Shpilka & Yehudayoff, 2010). Second, arithmetic circuit search provides a controlled testbed for learning symbolic search heuristics, analogous to the role played by SAT solving as a benchmark for combinatorial search and solver design (SAT; Kurin et al., 2020; Yolcu & Póczos, 2019). Finally, the simplicity of the arithmetic circuit environment makes it a useful model problem for more complicated sequential circuit-construction tasks, including quantum circuit synthesis and exact proof search in Lean (Moro et al., 2021; Fösel et al., 2021; Yang et al., 2023; Xin et al., 2025).

2. The Arithmetic Circuit Problem

In this section, we describe the two formulations of the arithmetic circuit problem that we considered. We introduce the concept of a `FactorLibrary` and discuss its implementation under both formulations.

2.1. Background

Let $\mathbb{F}_p$ be a finite field and let $X = \{x_0, \ldots, x_{n-1}\}$ be a set of variables. An arithmetic circuit over $\mathbb{F}_p[X]$ is a finite directed acyclic graph whose input gates are labeled by variables and constants and whose internal gates are labeled by addition $+$ or multiplication $\times$ (Bürgisser et al., 1997; Shpilka & Yehudayoff, 2010). The value at each internal gate is computed by performing the labeled operation on the values of the two incoming gates, and one gate is designated as the output gate. A circuit C computes f if the value at the output gate is equal to f. The size (or complexity) of a circuit C is the total number of internal gates.

Problem 2.1 (The arithmetic circuit problem)**.** Given a target polynomial $f \in \mathbb{F}_p[X]$, find a circuit C with the minimum number of internal gates such that C computes f.

The arithmetic circuit problem is central to algebraic complexity theory. It is closely tied to Valiant's VP vs VNP conjecture, which is the algebraic analogue of P vs NP (Valiant, 1979b). Finding efficient circuits for VNP-class polynomials such as permanents would be of great interest (Valiant, 1979a). Even though this problem is hard in general, there are certain families of polynomials for which well-known efficient circuits exist. Horner's method (Horner, 1819) computes any degree-d univariate polynomial using $2d$ gates, which is known to be optimal. Elementary symmetric polynomials and determinants satisfy similar recursions that produce efficient circuits. Moreover, a polynomial admits shorter-than-average circuits if it contains multiple factorizable subexpressions. Therefore, sums of products of polynomials are also a good heuristic for efficient circuits. For more, see Bürgisser et al. (1997).

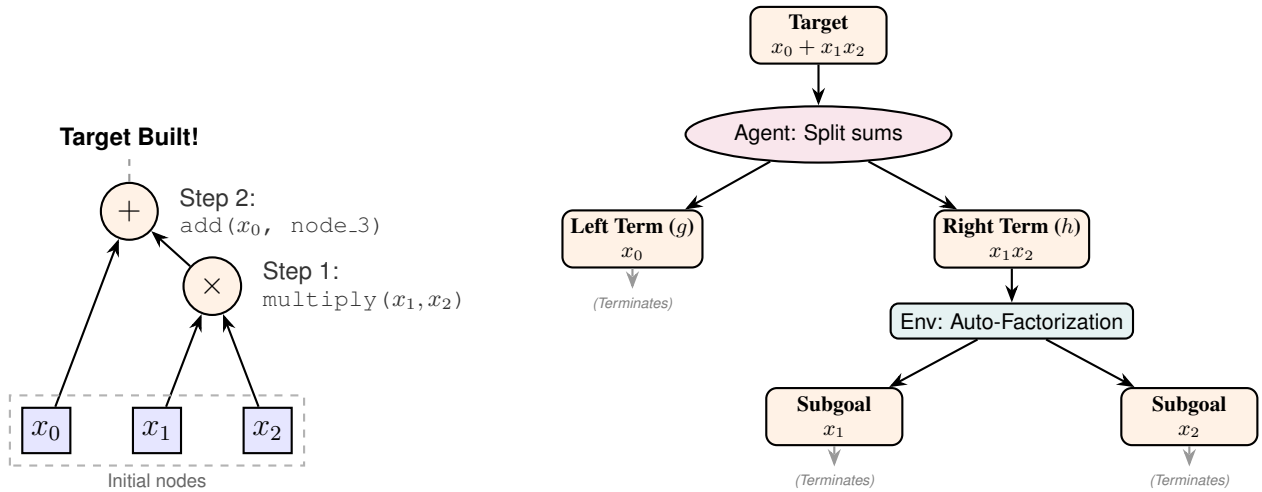
2.2. FactorLibrary and Recursive Subgoals

Polynomial arithmetic circuit optimization is closely related to algebraic expression optimization, program synthesis, library learning, and goal-directed reinforcement learning. Polynomial expressions can be optimized by combining algebraic factorization with common subexpression elimination (Hosangadi et al., 2006). More broadly, the idea of discovering reusable symbolic components has been studied in program synthesis and library learning. DreamCoder learns reusable abstractions across synthesis tasks through wake-sleep Bayesian program learning (Ellis et al., 2020), while LILO learns interpretable libraries by compressing and documenting code (Grand et al., 2024). These works show that reusable symbolic components can improve search efficiency while preserving interpretability.

Our `FactorLibrary` follows a similar motivation, but adapts it to arithmetic circuit search. Instead of storing general program fragments, a `FactorLibrary` stores reusable polynomial factors or factorizable subexpressions that serve as subgoals across training episodes. Each library element is an explicit polynomial object, making the stored knowledge inspectable, verifiable, and directly meaningful in the target algebraic domain.

The `FactorLibrary` is also related to goal-directed and hierarchical reinforcement learning. Hierarchical Deep Reinforcement Learning uses intrinsic goals to decompose sparse-reward tasks (Kulkarni et al., 2016). Hindsight Experience Replay improves sparse-reward learning by relabeling achieved states as goals (Andrychowicz et al., 2017). Automatic Goal Generation builds curricula of achievable but informative goals (Florensa et al., 2018). In contrast to these more general subgoal-generation methods, our `FactorLibrary` provides algebraically meaningful intermediate targets rather than learned latent goals or relabeled states.

It can be adapted to both the bottom-up and top-down formulations, although implementation details differ for each formulation. In both cases, the library acts as a form of symbolic memory: it records useful algebraic structures encountered or constructed during search and provides intermediate targets that guide the agent toward efficient circuit constructions. Compared to other subgoal techniques, the `FactorLibrary` has the advantage that each member is



(a) Bottom-up Environment

(b) Top-down Environment

Figure 1. Comparison of RL environment mechanics for finding arithmetic circuits for the target polynomial $f = x_0 + x_1x_2$.

an honest polynomial, so the subgoals are fully explicit, interpretable, and verifiable.

2.3. The Bottom-Up Formulation

We adopt the formulation of the arithmetic circuit problem from Zhang et al. (2026), which we refer to as the *bottom-up* approach. Given a polynomial $f(x_0, \ldots, x_{n-1})$, the initial game state consists of the variable gates $x_0, \ldots, x_{n-1}$ and a constant gate 1. Each subsequent game state at the mth iteration then consists of a partial circuit with gates $v_0, \ldots, v_{m-1}$. A valid action $\star \in \{+, \times\}$ connects two existing nodes v_i, v_j and produces a new node $v_m = v_i \star v_j$. An episode succeeds when some node computes f. As the number of variables and the circuit size grows, the action space grows exponentially (see Zhang et al. (2026, Section 3.2)), and non-optimized sequences of actions have very little chance of building a circuit which computes the target polynomial. Therefore, the bottom-up formulation is characterized by a sparse reward signal with large combinatorial growth. See Figure 1 for a schematic of the bottom-up environment.

In the bottom-up formulation, we implement the `FactorLibrary` as an iteration-level short-term memory that stores reusable subexpressions. At the start of each episode, the target f is factorized over $\mathbb{Z}$ using `SymPy`'s `factor_list` and then reduced mod p. All nontrivial polynomial factors, those of degree at least one that are not base input nodes, are collected into the active subgoal set $\mathcal{F}_f$. During rollout, the first time a newly appended gate u matches a member of $\mathcal{F}_f$, the agent receives a subgoal reward. If that subgoal is also present in the cross-episode library, an additional library bonus is awarded to encourage rediscovery of previously learned subexpressions.

We add two further kinds of reward signal to the library. Whenever u is the newly added node, the residual $f - u$ is computed immediately and added to the library. In the case that f is divisible by u, the quotient f/u is added to the library. One-time completion bonuses are given when the circuit simultaneously contains u and a node v' satisfying $u + v' = f$ or $u \cdot v' = f$. Each bonus is given at most once per episode to ensure the primary success criterion remains exact target construction. After a successful episode, every non-input node of the circuit is entered into the library for future episodes.

2.4. The Top-Down Formulation

We next introduce the *top-down* formulation, which reverses the direction of search and regards the arithmetic circuit problem as a recursive decomposition problem. The initial state simply consists of the list (f) where f is the target polynomial. The valid actions are all two-part decompositions of the form $f = g + h$ for polynomials g, h. We refer to these as *addition split points*. Once a split point is chosen, the game environment then immediately factors g and h and returns a list of factors $(g_1, \ldots, g_r,\ h_1, \ldots, h_s)$, and this list together with the chosen addition split point and multiplications in the factorizations becomes the next state. For each subsequent state $(f_1, \ldots, f_l)$, the valid actions are addition split points for all factors, in the form $f_i = g_i + h_i$. The game terminates when the list consists of only single variables x_j and constants 1. See Figure 1 for a schematic of the top-down environment.

One advantage of this formulation lies in its deterministic nature. The agent is not asked to guess the next gate in a partial circuit. Instead, the agent chooses how to decompose the remaining target. Since factorization algorithms over

finite fields are exact, the agent is always *guaranteed* to build a valid circuit for f given enough action steps.

In the top-down formulation, the `FactorLibrary` stores factorizable polynomials, specifically those for which building the polynomial via its factorization takes strictly fewer gates than direct construction. The library is prebuilt with examples such as $(x_i + a)^2$ for each variable x_i and each nonzero constant $a \in \mathbb{F}_p$, $(x_i + a)(x_j + b)$ for each variable pair and all constants $a, b \in \mathbb{F}_p$, and elementary symmetric polynomials up to degree three. As search proceeds, every factorizable polynomial is included, so the library grows dynamically with the agent's sequence of actions.

Before the agent chooses a split point on a polynomial f in the current state, the library identifies all entries g whose coefficients are strictly smaller than those of f, so that g can serve as an additive component. The agent gains a reward, up to a scaling factor, if it chooses an action that hits a library entry. See Figure 8b for an illustration.

Another advantage of the top-down environment is that it has a much smaller effective action space for fixed circuit complexity. If a size-C circuit has A addition gates and M multiplication gates, with $C = A + M$, then the bottom-up agent must search over all C gate choices. In contrast, the top-down agent only chooses additive split points, whereas multiplication gates are supplied by deterministic factorization and not part of the action sequence. The number of top-down action sequences is at most K^A where K is the total number of candidate splits, rather than the space of all circuits of complexity C. Since typically $A \leq C$ and often $A \ll C$ for factorizable targets, this gives a substantial reduction in effective search space. Indeed, this dramatic reduction in the size of the action space has led to superior training trajectories for the top-down agents compared to the bottom-up ones, as we discuss below.

3. Training

We now describe the training pipelines for the two formulations introduced in Section 2. For bottom-up, we adopted `CircuitBuilder`'s gate-action environment and added JAX-`mctx` Gumbel search, PPO updates, and `FactorLibrary` shaping. We only considered PPO, because Zhang et al. (2026) points out that PPO performs better at higher complexities. For top-down, we train a policy over variable-sized candidate decompositions, using factorization, memoization, `FactorLibrary` and AND/OR MCTS to evaluate recursive subgoals. We compared the efficacy of PPO and SAC in this new environment. All built polynomials are checked with exact arithmetic over $\mathbb{F}_p$. We summarize both training pipelines in Figure 7.

3.1. Bottom-Up Gumbel-PPO-MCTS

In the bottom-up environment, each partial circuit is represented as a DAG whose nodes store finite-field polynomials. Observations contain a padded circuit graph, node features, the target coefficient vector, and a valid-action mask. Actions are encoded by an operation $\star \in \{+, \times\}$ together with an upper-triangular pair of existing nodes (v_i, v_j), producing the new node $v_i \star v_j$. A graph encoder embeds the partial circuit, a target encoder embeds the coefficient vector of the target polynomial, and the resulting representation is passed to policy and value heads.

The implementation is built on JAX, with the network written in Flax and optimization handled by optax with gradient-clipped Adam. The environment is also implemented in pure JAX as a collection of fixed-size arrays (node coefficient matrix, edge arrays, subgoal arrays, and target vector), enabling JIT compilation of the full step function. All B parallel environments are stepped in a single JIT-compiled call, and the `FactorLibrary` reward path, including subgoal coefficient comparisons, library-known flag lookups, and completion checks, is executed as JAX operations over preloaded arrays, keeping the entire training loop GPU-resident. Batched MCTS uses Google DeepMind's `mctx` library, which searches all B environments simultaneously on GPU.

At each rollout step, the policy is locally improved by root-only Gumbel search (Danihelka et al., 2022). The network supplies masked logits and a value estimate. Gumbel-Top-k sampling selects a bounded set of promising root actions, and Sequential Halving compares those candidates by shallow lookahead. The search returns both a selected gate action and a completed-Q root distribution. PPO then updates the network using the clipped surrogate objective (Schulman et al., 2017), with advantages computed by generalized advantage estimation (Schulman et al., 2016). The stored network log-probability, not the MCTS probability, is used in the PPO ratio. The Gumbel-improved root distribution is used as an auxiliary cross-entropy distillation target, following the expert-iteration pattern in which search improves local decisions and the network generalizes these improved decisions across all states (Anthony et al., 2017; Silver et al., 2018).

The `FactorLibrary` operates on the level of reward shaping. At episode reset, nontrivial factors of the remaining target are added as episode subgoals. During rollout, we give a subgoal reward to a newly constructed gate if it is a first-time match for a library subgoal, and a library bonus if that subgoal was previously constructed in a successful episode. If the new node u and an existing node v satisfy $u + v = f$ or $uv = f$, the transition receives a completion bonus. These bonuses are bounded and one-time per episode, so the success criterion remains tied to exactly

Algorithm 1 Bottom-Up Gumbel-PPO-MCTS with `FactorLibrary`

Require: target distribution $\mathcal{D}_{\text{tar}}$, budget B, network (π_θ, V_θ), library $\mathcal{L}$, search budget K
for each training episode **do**
  Sample target $f \sim \mathcal{D}_{\text{tar}}$ and reset circuit $C \leftarrow \{1, x_0, \ldots, x_{n-1}\}$
  Initialize unrewarded subgoals $\mathcal{F}_f \leftarrow \text{NontrivialFactors}(f)$
  for $t = 0, \ldots, B-1$ **do**
    Encode $s_t = (C, f, \mathcal{F}_f, \mathcal{L})$ with valid gate-action mask
    $(a_t, \widehat{\pi}_t) \leftarrow \textsc{GumbelRootSearch}(s_t, \pi_\theta, V_\theta, K)$
    Append $u \leftarrow v_i \star v_j$ to C for $a_t = (\star, i, j)$
    Set r_t by exact terminal, subgoal, library, and one-step completion checks
    Remove any newly rewarded subgoals from $\mathcal{F}_f$
    Store $(s_t, a_t, r_t, \widehat{\pi}_t, V_\theta(s_t), \log \pi_\theta(a_t \mid s_t))$
    if $u = f$ **then**
      Add constructed non-input nodes of C to $\mathcal{L}$ and terminate episode
    end if
  end for
  Update θ by PPO with GAE and distillation to $\widehat{\pi}_t$
end for

Algorithm 2 Top-Down PPO+MCTS and SAC with `FactorLibrary`

Require: target distribution $\mathcal{D}_{\text{tar}}$, budget H, model (π_θ, V_θ), optional critic Q_ϕ, library $\mathcal{L}$, search budget K
for each training episode **do**
  Sample target $f \sim \mathcal{D}_{\text{tar}}$
  Reset list of subgoals $\mathcal{Q} \leftarrow (f)$, trace $\mathcal{T} \leftarrow \emptyset$, memo table $M \leftarrow \emptyset$
  for $t = 0, \ldots, H-1$ **do**
    if $\mathcal{Q} = \emptyset$ **then**
      Reconstruct and verify circuit from $\mathcal{T}$; add useful subtraces to $\mathcal{L}$
      Terminate episode
    end if
    Let q be the active unresolved polynomial
    Generate verified actions $\mathcal{A}(q)$: splits, library matches, and possible factor action
    $(a_t, \widehat{\pi}_t) \leftarrow \textsc{AndOrMCTS}(q, \mathcal{A}(q), M, \pi_\theta, V_\theta, K)$
    Execute a_t: factor selected pieces, charge rebuild cost, and update $\mathcal{Q}, \mathcal{T}, M$
    Set reward to baseline cost saving plus any verified library reward
    Store transition and search target $\widehat{\pi}_t$ in rollout or replay buffer
  end for
  Update by PPO with MCTS distillation, or by SAC from replay using candidate-set Q-values
end for

constructing the target. The `FactorLibrary`, however, persists across all episodes and serves as the short-term memory of the model. See Algorithm 1 for pseudocode, Figure 1a and Figure 8a for illustrations.

3.2. Top-Down PPO+MCTS and SAC

The top-down implementation trains on the decomposition environment described in Section 2.4. At each decision point, the active polynomial is the first element in the list of unresolved subgoals. The environment generates a variable-sized candidate set consisting of additive splits and, when the active polynomial factors nontrivially, a whole-polynomial factor action. Candidate splits are generated from term supports, random masks, common-factor clusters, known patterns, and `FactorLibrary` matches. The policy scores the current candidate list generated by the environment.

After an action is selected, the environment factors the split pieces over $\mathbb{F}_p$. Only additive splits are counted towards agent actions and penalized in reward shaping. Base cases and memoized subproblems are resolved immediately, and unresolved nontrivial factors are pushed back onto the game state. The step reward is the reduction of cost before and after the action compared to the baseline cost given by random actions and heuristic search, with an optional library-match bonus.

Planning is performed by memoized AND/OR PUCT search, building on UCT and MCTS (Kocsis & Szepesvári, 2006; Browne et al., 2012). OR nodes are candidate actions for a polynomial. AND expansions are deterministic: after an action is chosen, factorization determines all child subproblems that must be solved. Search backs up total circuit cost rather than the value on a single leaf node. Our use of priors and values follows the AlphaZero search regime (Silver et al., 2018).

In PPO+MCTS mode, MCTS runs at each action step. The distribution of root visits is used for action selection and as a cross-entropy distillation target in the PPO loss. We follow Schulman et al. (2017) for clipped surrogate, value loss, and entropy bonus. In SAC mode, the critic scores each candidate decomposition action by its expected entropy-regularized return, and the actor is trained with a discrete-action maximum-entropy objective (Haarnoja et al., 2018; Christodoulou, 2019). See Algorithm 2 for pseudocode.

4. Results and Comparisons

In this section, we discuss our training and evaluation results. For the bottom-up branch, we analyze the success rate of building a circuit for the target polynomial across the training curriculum, which is based on increasing heuristic size of the circuits. For the top-down branch, we analyze the success rate of the agent beating the baseline and achieving the optimal circuit length across circuit sizes.

4.1. Bottom-Up Gumbel-PPO-MCTS

The bottom-up Gumbel-PPO-MCTS run uses a JAX implementation with `mctx` Gumbel MuZero search. `FactorLibrary` provides reward shaping. We trained using a curriculum based on circuit size, over two-variable targets in $\mathbb{F}_5$. Let C_i denote circuits of heuristic size i, meaning circuits generated from i random gates. The curriculum proceeds through adjacent complexity buckets, starting from C_1, then C_1, C_2, then C_2, C_3, and continuing up to C_9, C_{10}. We trained the agent using a 4-layer GNN with hidden dimension 384, batch size 1024, `mctx` Gumbel MuZero search with 64 simulations and 16 actions, on a single NVIDIA RTX PRO 6000 Blackwell GPU. A condensed summary of the curriculum is given in Table 1; the full phase-by-phase results are in Table 3 in the appendix.

The results show that Gumbel search and the `FactorLibrary` make the learning effective at low circuit size. The agent had 99.2% aggregate success up to C_3, showing that the network has memorized the constructions of most small-size circuits. Performance remains strong through C_4, but begins to degrade at C_5, and the drop becomes sharper at higher complexities. The C_5, C_6 phase ends at 45.0%, the C_6, C_7 phase at 26.8%, and the C_9, C_{10} phase at 7.8%. This is the expected failure mode of the primitive gate-action formulation, which coincides with the combinatorial explosion of the search space. Even with improved root search and factor-library rewards, the agent still struggles to discover useful intermediate expressions inside a rapidly expanding action space.

Table 1. Bottom-up Gumbel-PPO-MCTS curriculum (condensed). Seven representative phases illustrate the learning trajectory. Full results in Table 3.

Phase	Train buckets	Agg. success
1	C_1	93.7%
3	C_2, C_3	99.2%
4	C_3, C_4	91.7%
5	C_4, C_5	71.7%
6	C_5, C_6	45.0%
7	C_6, C_7	26.8%
10	C_9, C_{10}	7.8%

4.2. Top-Down PPO+MCTS versus SAC

We trained the top-down PPO+MCTS agent with an MLP with 3 hidden layers of width 128 on an ensemble of targets combining Horner schemes, sums of products, and elementary symmetric polynomials. The training ran for 1000 iterations and used 16 rollouts per update, 16 candidate actions per step, maximum episode length 24, and 48 MCTS simulations of depth 6 on an RTX 4070 Ti. The top-down SAC agent used the same actor architecture as PPO but replaced MCTS with off-policy discrete SAC using twin 3-layer width-128 critics, replay batch size 64, 16 gradient steps per iteration, Polyak target updates, and an automatically tuned entropy temperature with target entropy scale 0.4.

For evaluation, we used a fixed 450-target training set for in-distribution tracking and a disjoint 207-target held-out set for the reported generalization results, across $C_2, \ldots, C_{10}$. For $C_2, \ldots, C_8$, the C_k labels are computed by exact search over the split/factor/direct move set. The C_9 and C_{10} buckets are partly heuristic-labeled, so verified-optimality claims are limited to $C_2, \ldots, C_8$. See Table 2 for success rates and Figure 6 for training curves.

Table 2. Top-down evaluation metrics.

Method	Final	Best	Train	Gap	Time
PPO+MCTS	0.918	0.918	0.918	0.01	2.5 h
SAC	0.889	0.928	0.844	0.05	30 m
Random	0.40	–	–	–	–

Both top-down methods substantially outperform the random baseline. PPO+MCTS reaches a final held-out C_k-match rate of 0.918 ($k \leq 8$), compared with 0.40 for uniform random action selection over the same candidate sets. This indicates that the learned split/factor policy generalizes to unseen targets rather than memorizing the training set. SAC is competitive despite not using MCTS. Its final held-out success rate is 0.889, and its best checkpoint reaches 0.928, slightly above the final PPO+MCTS checkpoint. The SAC method is much cheaper computationally, as each run takes about 30 minutes on an RTX 4070 Ti, compared with about 2.5 hours for PPO+MCTS. However, it appears less stable due to drift late in training. The choice between the two methods is therefore stability versus compute.

4.3. Limitations

The scale of our experiments is restricted to few variables, coefficients in $\mathbb{F}_5$, and relatively small circuit complexity. In the bottom-up environment, the action space grows quadratically in the number of variables n and complexity C, and the total search space grows super-exponentially in n and C (Zhang et al., 2026). The top-down action space can be understood in terms of the number of split points. An active polynomial of degree at most d in n variables over $\mathbb{F}_p$ admits $p^{\binom{n+d}{n}}$ ordered additive decompositions $f = g + h$, since g ranges freely over that space and the degree satisfies $d \leq 2^C$. The growth of the search space in both environments makes training lengthy, and our experiments are constrained by available computing resources. Additional work is necessary to extend our methods to highly complex polynomials.

5. Conclusion and Future Directions

We studied two reinforcement learning formulations for the arithmetic circuit problem and implemented `FactorLibrary` in each. The bottom-up formulation extends gate-level search with Gumbel-PPO-MCTS and `FactorLibrary` reward shaping, implemented in JAX with `mctx` for GPU-batched MCTS. The top-down formulation searches over recursive additive split points, using memoized AND/OR MCTS and a JAX-accelerated `FactorLibrary` to produce candidate actions.

Our results show that top-down agents perform substantially better at higher circuit complexities. The challenge for bottom-up is that MCTS search performance degrades sharply beyond C_5, consistent with the combinatorial explosion of the action space. The top-down agents, which are fed algebraic subgoals directly rather than individual gates, maintain strong performance through C_{10}. The SAC variant achieves competitive results at a fraction of the compute, suggesting that the top-down environment is well suited for off-policy learning. The `FactorLibrary` can be well-adapted to both formulations. It stores reusable subgoals that are certified and fully interpretable, enabling sophisticated reward shaping across multiple episodes.

In future work, we plan to scale to more variables and higher degree targets, more carefully separate training-time from inference-time compute, and identify which polynomial families favor each formulation. We hope to use our trained models to discover efficient circuits for permanents and other VNP-hard families of polynomials. In a follow-up work, we will study a two-player formulation of the top-down environment. The `FactorLibrary` mechanism also alludes to a domain-specific analogue of a "lemma library" in the context of auto-proof generation. The arithmetic circuit problem provides a controlled setting for studying how learned systems acquire, reuse, and generalize symbolic subgoals.

Acknowledgements

This project is a part of the UW Math AI Lab. We thank the UW eScience School for computing resources. CPU and GPU computing were in part done using AWS credits from the UW eScience School and UW IT, and also in part done using the UW Research Computing Club funded from the UW Student Technology Fee Committee. Parts of the code base and diagrams in the paper were produced with the help of ChatGPT 5.5 and Claude Opus 4.7.

References

SAT Competition. https://satcompetition.github.io/. Accessed 2026-06-19.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Pieter Abbeel, O., and Zaremba, W. Hindsight experience replay. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

Anthony, T., Tian, Z., and Barber, D. Thinking fast and slow with deep learning and tree search. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. A survey of Monte Carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.

Bürgisser, P., Clausen, M., and Shokrollahi, M. A. *Algebraic complexity theory. With the collaboration of Thomas Lickteig*, volume 315 of *Grundlehren Math. Wiss.* Berlin: Springer, 1997. ISBN 3-540-60582-7.

Christodoulou, P. Soft actor-critic for discrete action settings. *arXiv preprint arXiv:1910.07207*, 2019.

Danihelka, I., Guez, A., Schrittwieser, J., and Silver, D. Policy improvement by planning with Gumbel. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=bERaNdoegnO.

Ellis, K., Wong, C., Nye, M., Sablé-Meyer, M., Cary, L., Morales, L., Hewitt, L. B., Solar-Lezama, A., and Tenenbaum, J. B. Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning. *arXiv preprint arXiv:2006.08381*, 2020.

Florensa, C., Held, D., Geng, X., and Abbeel, P. Automatic goal generation for reinforcement learning agents. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 1515–1528. PMLR, 2018. URL https://proceedings.mlr.press/v80/florensa18a.html.

Fösel, T., Niu, M. Y., Marquardt, F., and Li, L. Quantum circuit optimization with deep reinforcement learning, 2021. URL https://arxiv.org/abs/2103.07585.

Grand, G., Wong, L., Bowers, M., Olausson, T. X., Liu, M., Tenenbaum, J. B., and Andreas, J. Lilo: Learning interpretable libraries by compressing and documenting code. In *International Conference on Learning Representations*, 2024.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018. URL https://arxiv.org/abs/1801.01290.

Horner, W. G. Xxi. a new method of solving numerical equations of all orders, by continuous approximation. *Philosophical Transactions of the Royal Society of London*, 109:308–335, 12 1819. ISSN 0261-0523. doi: 10.1098/rstl.1819.0023. URL https://doi.org/10.1098/rstl.1819.0023.

Hosangadi, A., Fallah, F., and Kastner, R. Optimizing polynomial expressions by algebraic factorization and common subexpression elimination. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(10):2012–2022, 2006. doi: 10.1109/TCAD.2006.875712.

Hubert, T., Mehta, R., Sartran, L., Horváth, M. Z., Žužić, G., Wieser, E., Huang, A., Schrittwieser, J., Schroecker, Y., Masoom, H., Bertolli, O., Zahavy, T., Mandhane, A., Yung, J., Beloshapka, I., Ibarz, B., Veeriah, V., Yu, L., Nash, O., Lezeau, P., Mercuri, S., Sönne, C., Mehta, B., Davies, A., Zheng, D., Pedregosa, F., Li, Y., von Glehn, I., Rowland, M., Albanie, S., Velingker, A., Schmitt, S., Lockhart, E., Hughes, E., Michalewski, H., Sonnerat, N., Hassabis, D., Kohli, P., and Silver, D. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, nov 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09833-y. URL https://doi.org/10.1038/s41586-025-09833-y.

Kocsis, L. and Szepesvári, C. Bandit based monte-carlo planning. In *European Conference on Machine Learning*, pp. 282–293. Springer, 2006.

Kulkarni, T. D., Narasimhan, K., Saeedi, A., and Tenenbaum, J. B. Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

Kurin, V., Godil, S., Whiteson, S., and Catanzaro, B. Can q-learning with graph networks learn a generalizable branching heuristic for a sat solver? In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Lidl, R. and Niederreiter, H. *Finite Fields*, volume 20 of *Encyclopedia of Mathematics and Its Applications*. Cambridge University Press, Cambridge, 2 edition, 1997. ISBN 9780521392310.

Moro, L., Paris, M. G. A., Restelli, M., and Prati, E. Quantum compiling by deep reinforcement learning. *Communications Physics*, 4(178), 2021. doi: 10.1038/s42005-021-00684-3. URL https://doi.org/10.1038/s42005-021-00684-3.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*, 2016.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Shpilka, A. and Yehudayoff, A. Arithmetic circuits: A survey of recent results and open questions. *Foundations and Trends in Theoretical Computer Science*, 5(3-4):207–388, 12 2010. ISSN 1551-305X. doi: 10.1561/0400000039. URL https://doi.org/10.1561/0400000039.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T., Simonyan, K., and Hassabis, D. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362 (6419):1140–1144, 2018.

Valiant, L. The complexity of computing the permanent. *Theoretical Computer Science*, 8(2):189–201, 1979a. ISSN 0304-3975. doi: 10.1016/0304-3975(79)90044-6.

Valiant, L. G. Completeness classes in algebra. In *Proceedings of the Eleventh Annual ACM Symposium on Theory of Computing*, STOC '79, pp. 249–261, New York, NY, USA, 1979b. Association for Computing Machinery. ISBN 9781450374385. doi: 10.1145/800135.804419. URL https://doi.org/10.1145/800135.804419.

Xin, H., Ren, Z., Song, J., Shao, Z., Zhao, W., Wang, H., Liu, B., Zhang, L., Lu, X., Du, Q., Gao, W., Zhang, H., Zhu, Q., Yang, D., Gou, Z., Wu, Z., Luo, F., and Ruan, C. Deepseek-prover-v1.5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=I4YAIwrsXa.

Yang, K., Swope, A. M., Gu, A., Chalamala, R., Song, P., Yu, S., Godil, S., Prenger, R., and Anandkumar, A. Leandojo: Theorem proving with retrieval-augmented language models. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=g7OX2sOJtn.

Yolcu, E. and Póczos, B. *Learning local search heuristics for boolean satisfiability*. Curran Associates Inc., Red Hook, NY, USA, 2019.

Zhang, W. K., Pandey, R., Mehta, B., Jin, K., Morato, N., Ganapule, A., Zeng, M. R., and Alper, J. Circuitbuilder: From polynomials to circuits via reinforcement learning, 2026. URL https://arxiv.org/abs/2603.17075.

A. Results and Plots from Experiments

Table 3 gives the complete phase-by-phase curriculum for the bottom-up Gumbel-PPO-MCTS run. Each phase is initialized from the previous checkpoint. Library size tracks the number of distinct polynomials accumulated in the `FactorLibrary` across episodes; the peak at Phase 7 (15,051 entries) reflects the transition from regimes where many circuits are solved (and thus registered) to regimes where the agent rarely succeeds. Figures 2–6 show training curves and final per-complexity breakdowns for the top-down agents.

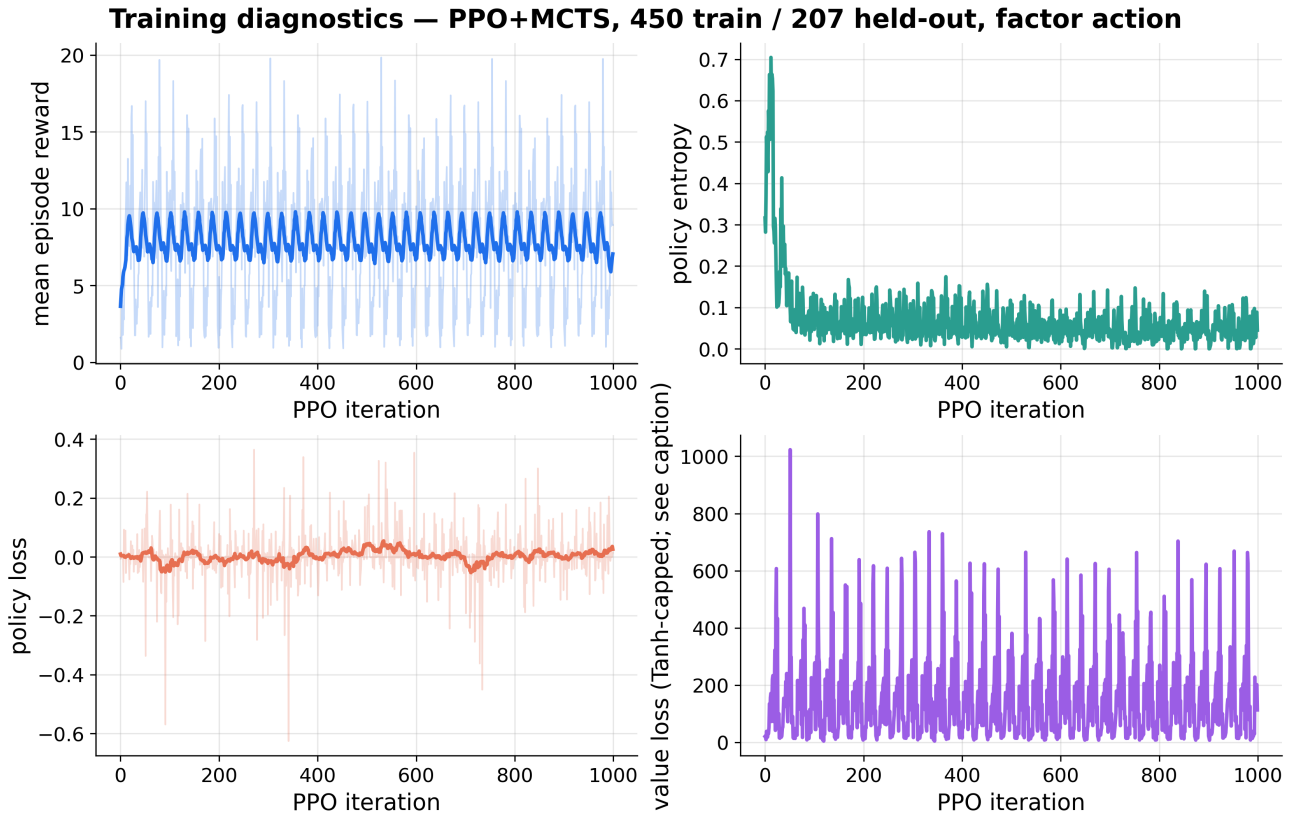


Figure 2. Training curves.

Table 3. Bottom-up JAX Gumbel-PPO-MCTS full curriculum. Each phase trains on the indicated complexity bucket(s) and is initialized from the previous phase. Success is exact target-construction rate in the gate-action environment.

Phase	Train Buckets	Iter.	Agg. Success	Per-Bucket Success	Lib. Size	Reward
1	C_1	30	93.65%	$C_1 = 93.7\%$	501	18.535
2	C_1, C_2	500	99.90%	$C_1 = 100.0\%,\ C_2 = 99.8\%$	959	19.263
3	C_2, C_3	500	99.22%	$C_2 = 100.0\%,\ C_3 = 98.4\%$	1368	18.696
4	C_3, C_4	500	91.70%	$C_3 = 100.0\%,\ C_4 = 83.4\%$	3985	17.240
5	C_4, C_5	500	71.68%	$C_4 = 89.6\%,\ C_5 = 53.7\%$	8839	13.137
6	C_5, C_6	500	45.02%	$C_5 = 59.6\%,\ C_6 = 30.5\%$	13693	7.973
7	C_6, C_7	500	26.76%	$C_6 = 35.2\%,\ C_7 = 18.4\%$	15051	4.686
8	C_7, C_8	500	17.19%	$C_7 = 22.7\%,\ C_8 = 11.7\%$	11625	2.786
9	C_8, C_9	500	12.40%	$C_8 = 16.0\%,\ C_9 = 8.8\%$	8860	1.830
10	C_9, C_{10}	500	7.81%	$C_9 = 8.4\%,\ C_{10} = 7.2\%$	7279	0.925

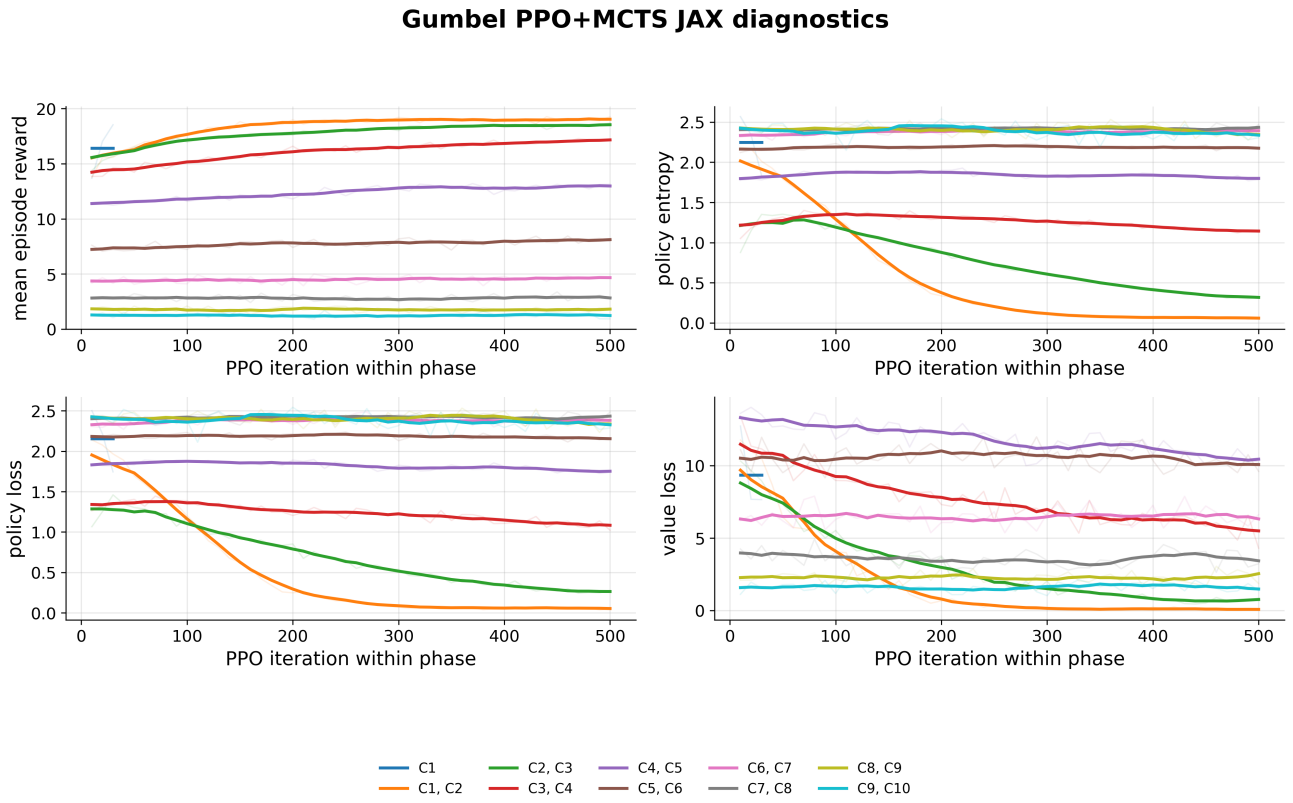


Figure 3. Overlaid diagnostics for the Gumbel curriculum phases.

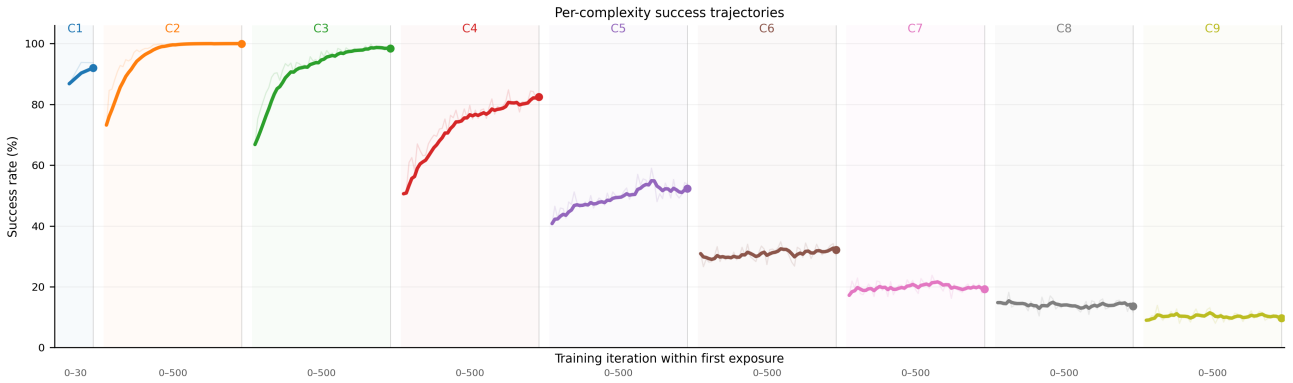


Figure 4. Staggered success rates across curriculum phases.

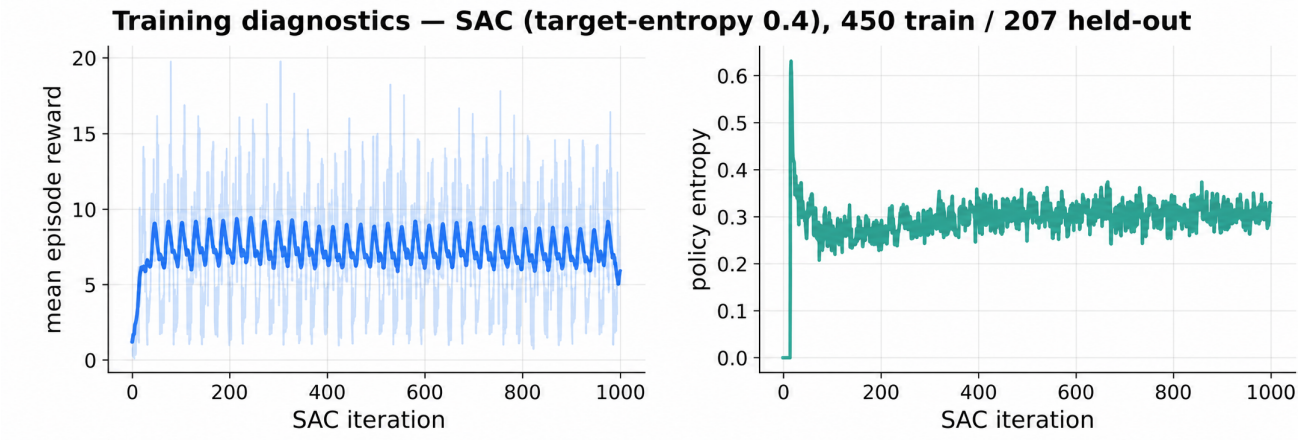


Figure 5. SAC training diagnostics for the top-down run.

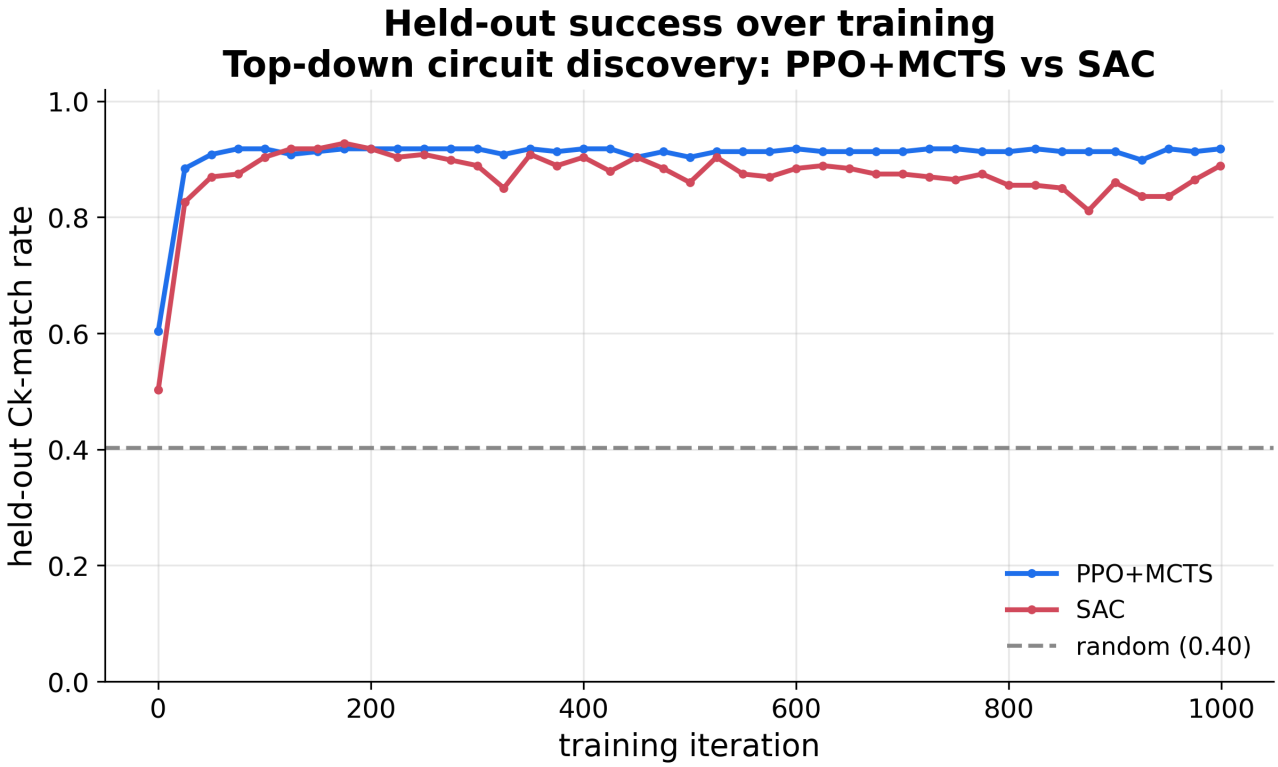
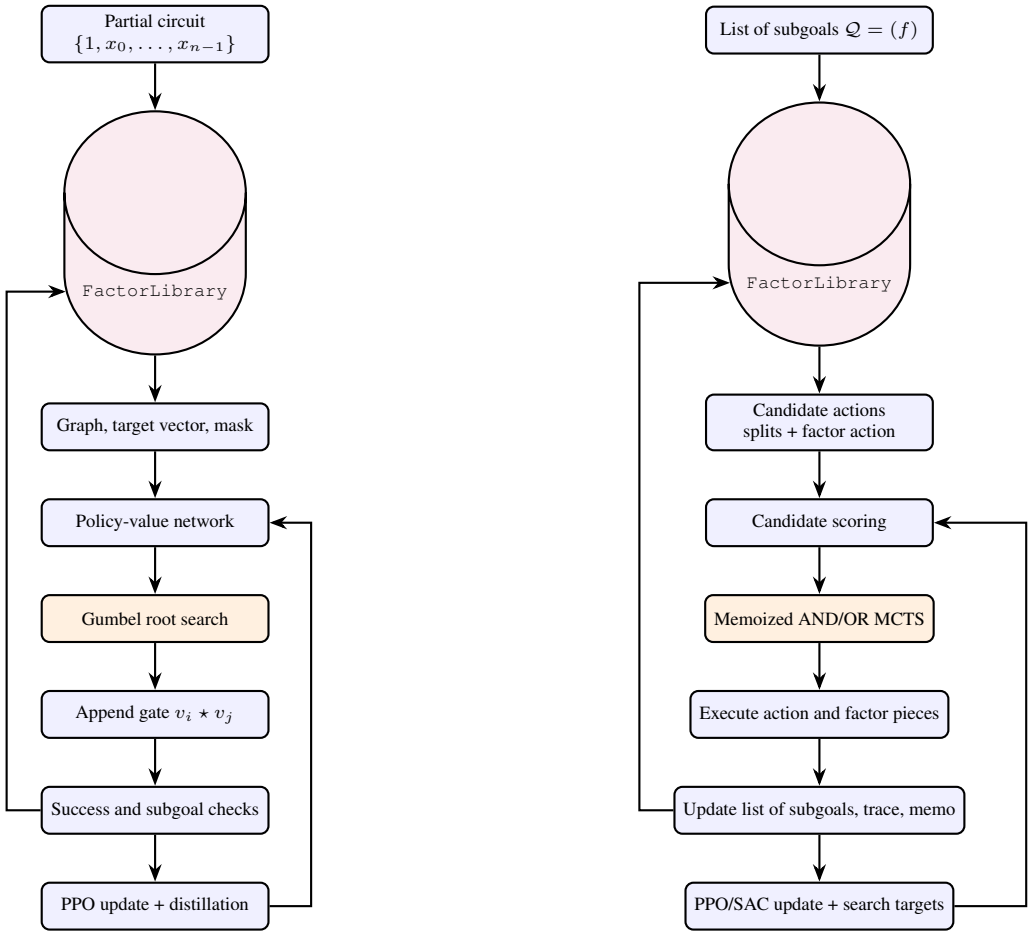
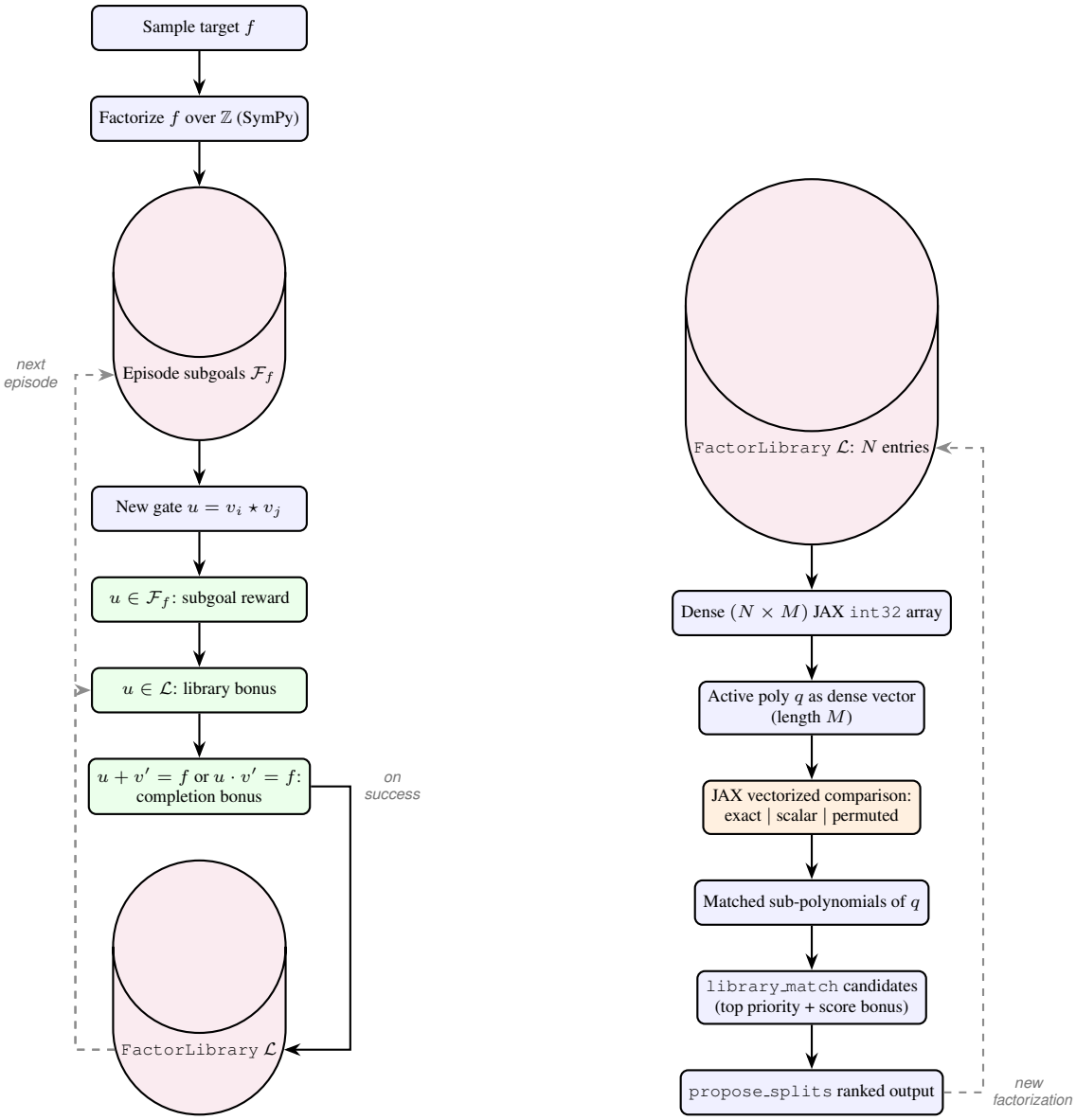


Figure 6. Held-out C_k-match rate over training iterations.



(a) Bottom-up pipeline. *(b)* Top-down pipeline.

Figure 7. Training pipelines for bottom-up and top-down environments.



(a) Bottom-up `FactorLibrary`. *(b)* Top-down `FactorLibrary`.

Figure 8. `FactorLibrary` in the two formulations.