# CIRCUITPILOT: LEARNING CIRCUIT OPTIMIZATION THROUGH IMITATION AND SELF-PLAY

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Optimizing a circuit means searching a large space of equivalent circuits for a cheaper one, and a bounded optimizer must choose which rewriting and search procedures to run. We propose CircuitPilot, which learns these choices in two stages: a policy first imitates the optimal first actions found by exhaustive short-horizon search, then improves through single-player self-play on targets from policy-guided tree search. We apply one formulation to polynomial, Boolean, and quantum circuits, with a separate policy per domain; every returned circuit is checked for equivalence to its input. On 161 three-qubit development circuits, the quantum policy matches or improves on the cost of a step-limited Quasar configuration on 152 (70 strictly better). An imitation-only polynomial policy obtains lower total cost than each of three fixed schedules, though not lower than their per-input best, and 98 of 100 IWLS 2026 Boolean benchmarks can be encoded as model inputs.

## 1 INTRODUCTION

A circuit computes a function by composing elementary gates, with wires carrying each gate's output to the inputs of later gates. Circuit synthesis constructs a circuit for a specified computation from a given gate set; a central goal is low cost, typically measured by gate count or by depth, the longest chain of dependent gates.

We study circuit optimization, which seeks a cheaper circuit equivalent to a given one, for three kinds of circuits. Polynomial circuits compute polynomials using addition and multiplication, Boolean circuits compute functions of bits using logic gates, and quantum circuits transform quantum states using gates acting on quantum bits (qubits). Each kind has an environment that provides domain-specific optimization procedures and equivalence checks.

Local rewrites replace a subcircuit with an equivalent subcircuit. These rewrites can expose shared computations or bring canceling gates together, but a rewrite sequence that ends in a cheaper circuit need not reduce cost at every step (Li et al., 2024). Equality saturation avoids committing to any single rewrite: it stores every equivalent expression it discovers in an e-graph, a data structure that groups equivalent expressions, and later extracts a low-cost circuit from them (Willsey et al., 2021). As the graph grows, time and memory limits constrain the number of alternatives explored. A bounded optimizer must therefore choose which procedures to run, such as rewriting rounds, factorization, subcircuit resynthesis, or extraction, and in what order.

We propose CircuitPilot, a framework for learning these procedure choices through imitation learning followed by reinforcement learning (RL). The policy observes the current circuit and the alternatives retained by the symbolic solver, then selects a procedure such as polynomial factorization or subcircuit resynthesis. We use the same decision-process formulation across the three environments, with a separate policy for each.

We train the policy in two stages. First, a symbolic teacher exhaustively searches all procedure sequences up to a short horizon and labels the first actions that lead to the lowest final cost; the policy imitates these labels. We then refine the policy through single-player self-play: the network's current policy and value estimates guide tree search over longer procedure sequences, producing improved action distributions as policy targets and realized cost reductions as value targets. At



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

deployment, the trained policy selects each procedure with one network evaluation, without tree search.

The quantum policy, trained by imitation and self-play, obtains lower circuit cost than a bounded configuration of the equality-saturation optimizer Quasar (Yang et al., 2026b) on 70 of 161 development circuits and equal cost on 82. The polynomial policy uses imitation alone and obtains lower total cost than each of three fixed procedure sequences (0.25% below the best), although running all three and selecting their lowest-cost circuits gives a lower total. For Boolean circuits, we test only encoding: 98 of 100 IWLS 2026 benchmarks (IWLS 2026 Programming Contest Organizers, 2026) can be loaded and encoded within the model's input limits.

## 2 Circuit optimization environments

An input circuit $C_0$ specifies the computation to preserve. Each domain $d$ defines an equivalence relation between circuits, $C \equiv_d C_0$, and a circuit cost $c_d(C)$. Costs are tuples compared lexicographically: later components only break ties in earlier ones. Table 1 gives these priorities. Each gate is counted once, including gates whose outputs are used more than once.

Table 1: Cost objectives and equivalence checks. Costs are compared from left to right. $N$ counts gates of the indicated type ($1q$: single-qubit), $D$ is depth, and $S$ counts every addition and each multiplication between nonconstant polynomials. Section 2.3 describes the derivation checks.

| Domain | Cost tuple | Circuit equivalence check |
|---|---|---|
| Polynomial | $(S, N_\times, N_+, D, N_{\text{total}})$ | Polynomial coefficients |
| Boolean | $(N_{\text{AND}}, N_{\text{XOR}}, D, N_{\text{total}})$ | Output truth tables |
| Quantum | $(N_{\text{CX}}, N_{\text{total}}, D, N_{1q})$ | Unitary operators or derivations |

The environments retain rewrite alternatives in e-graphs. An e-graph groups equivalent subexpressions into classes and records how operations combine those classes. A rewrite adds an equivalent expression without discarding the original. *Extraction* selects a concrete circuit from these alternatives according to the cost function. Even exact extraction minimizes cost only over circuits represented in the graph; further rewriting can expose a cheaper one.

### 2.1 Polynomial circuits

The polynomial environment represents polynomials with addition and multiplication gates over a finite prime field $\mathbb{F}_p$. Two polynomial circuits are equivalent when corresponding outputs compute identical formal polynomials. The checker compares their coefficients exactly; evaluating at field elements would not suffice, since $x^p$ and $x$ agree on every element of $\mathbb{F}_p$ but are distinct polynomials.

The primary objective minimizes the *scalar-free* operation count: every addition and each multiplication between nonconstant polynomials. Multiplication by a field constant is excluded from this count but included in the secondary gate counts. In Figure 1, factoring an expanded quadratic reveals a square already computed for another output. Reusing that square reduces the circuit from ten gates to four.

The policy chooses among algebraic rewriting, factorization, and circuit extraction procedures. Rewriting adds alternative polynomial expressions; factorization searches for decompositions that can share intermediate results. One procedure, the factorization portfolio, keeps several such decompositions so that extraction can choose among them jointly. Appendix B lists the available procedures and how shared intermediate values are stored.

### 2.2 Boolean circuits

The Boolean environment represents circuits as XOR-AND graphs (XAGs): gates compute exclusive-or and conjunction, and edges may carry a negation. Negation has no gate cost. Two Boolean circuits are equivalent if their corresponding outputs have identical truth tables.

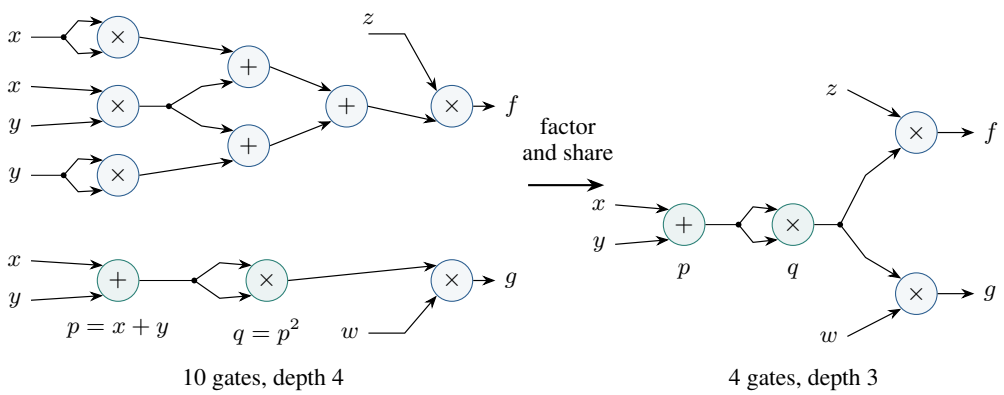


Figure 1: Factoring exposes a square that can be shared across outputs. Initially $f = z[(x^2 + xy) + (xy + y^2)]$ and $g = w(x + y)^2$. Local factoring gives $x^2 + xy = x(x + y)$ and $xy + y^2 = y(x + y)$, then $x(x + y) + y(x + y) = (x + y)^2$. Both outputs can use the existing $q = p^2$. The $xy$ product on the left is already shared. Repeated input labels denote the same variable; all gates are binary, and a square uses one multiplication.

The objective minimizes the number of AND gates, with XOR count and depth breaking ties. In Figure 2, factoring a full adder's carry expression lets it reuse an XOR from the sum circuit, saving one AND and one XOR gate.

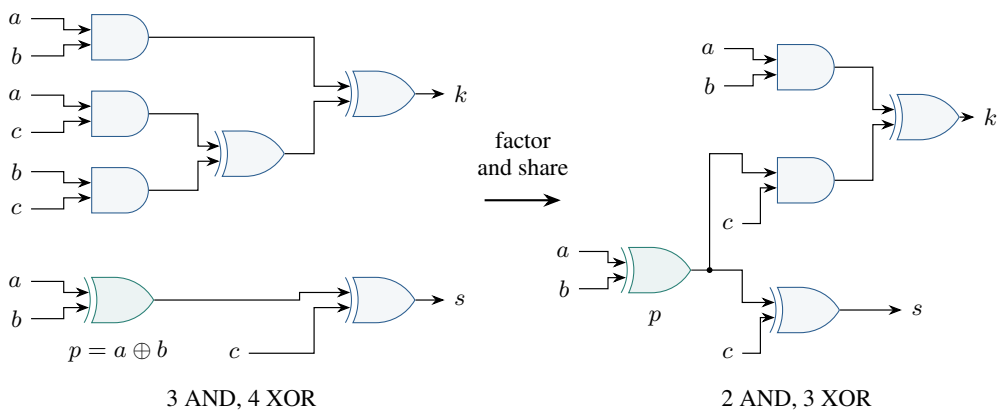


Figure 2: A full adder with sum $s = a \oplus b \oplus c$ and carry $k = (a \wedge b) \oplus (a \wedge c) \oplus (b \wedge c)$. Factoring $(a \wedge c) \oplus (b \wedge c) = c \wedge (a \oplus b)$ lets the carry reuse the sum circuit's $p = a \oplus b$, removing one AND and one XOR gate. The D-shaped gates are AND; the gates with a second input arc are XOR. Repeated input labels denote the same bit. Both circuits have depth 3.

The procedures rewrite expressions, *resynthesize* subcircuits (search for a cheaper replacement for a subcircuit with at most four boundary inputs), and extract a circuit from the e-graph. Extraction selects gates jointly across circuit outputs, so the shared XOR in Figure 2 is counted once even though both outputs use it. Appendix C specifies the network representation, extraction constraints, and encoding of large benchmark circuits.

### 2.3 Quantum circuits

Quantum circuits are represented as sequences of gates acting on qubit wires. The gate set is X (a bit flip), its square root SX, Z rotations (RZ) by rational multiples of $\pi$, and controlled-X (CX), which flips its target qubit when its control is one. The objective first minimizes CX count (the primary bottleneck in current quantum hardware), then total gate count and depth. In Figure 3, two rotations

of the same three-qubit parity merge: four CX gates cancel, allowing the rotations to fuse into one gate.

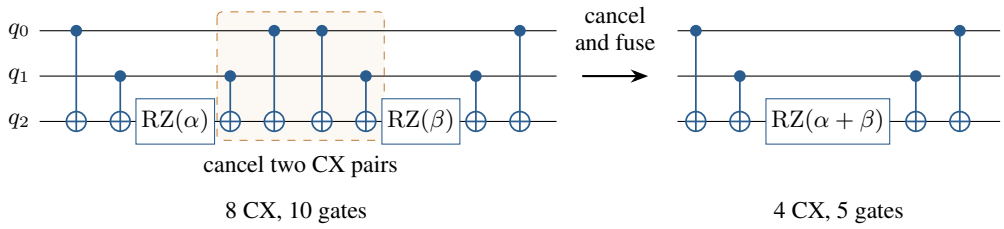


Figure 3: Combining two rotations of the same three-qubit parity. Each block computes $q_0 \oplus q_1 \oplus q_2$ on the third wire, applies RZ, and reverses the computation. Inside the dashed box, the two middle CX gates cancel, then the remaining two cancel. The rotations become adjacent and fuse by $\mathrm{RZ}(\alpha)\mathrm{RZ}(\beta) = \mathrm{RZ}(\alpha + \beta)$. Filled dots denote controls and circled pluses denote targets; gates act from left to right. The counts use $\alpha = \pi/4$ and $\beta = \pi/2$, so the fused rotation is nontrivial and remains one gate.

Two quantum circuits are equivalent when their unitary operators $U(C)$ agree up to global phase, $U(C') = e^{i\phi}U(C)$ for some $\phi$. For at most three qubits, the checker computes these operators using exact algebraic arithmetic (Appendix D). For circuits with more than three qubits, it checks a derivation from the input circuit to the candidate circuit using verified local replacements and reorderings that preserve wire dependencies. This verifies the executed transformation sequence without constructing the $2^n \times 2^n$ operator of an $n$-qubit circuit.

The quantum procedures rewrite gate sequences, resynthesize subcircuits, and extract circuits. A sequence e-graph stores equivalent gate-sequence prefixes and keeps distinct orderings of commuting gates, so a reordering can bring canceling gates together; a gate graph records gate input/output wires, merging orderings with the same wiring. A *rewrite–extract procedure* combines up to eight rewrite rounds with extraction and circuit rebuilding in a single policy decision. Appendix D specifies the two representations and the procedure budgets.

## 3 Learning from Symbolic Search

We formulate circuit optimization as a sequence of decisions about which symbolic procedure to execute. The policy first imitates the choices of exhaustive search over a short horizon. It then guides tree search over longer sequences and learns from the resulting trajectories. Figure 4 shows the training process and the policy used at deployment.

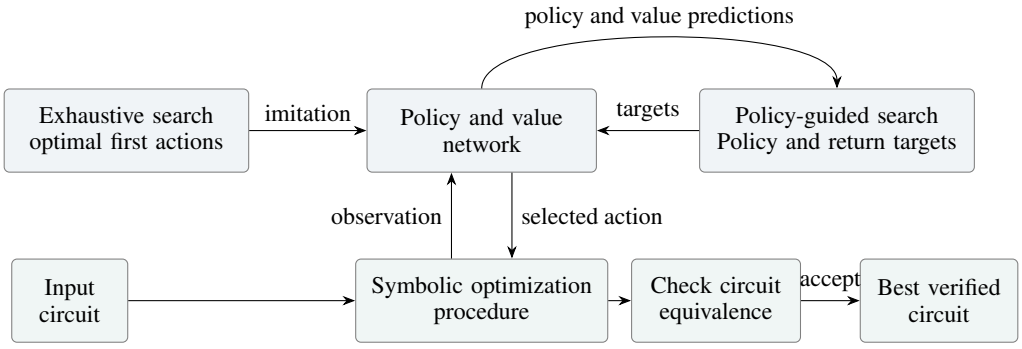


Figure 4: Short exhaustive searches supply optimal first actions for imitation learning; policy-guided searches supply policy and return targets for self-play. At deployment, the trained policy selects symbolic optimization procedures in the lower observation–action loop. A candidate circuit replaces the best circuit only after it is verified as equivalent to the input.

### 3.1 States, actions, and rewards

An action invokes an optimization procedure from Section 2. It is a *macro-action* because a procedure can apply several local rewrites or search for a replacement subcircuit. For example, polynomial factorization adds alternatives such as $x^2 + xy = x(x + y)$ in Figure 1; extraction selects a circuit implementing all outputs from the accumulated alternatives. A sequence of actions is a *schedule*.

The environment retains the lowest-cost circuit verified as equivalent to the input, called the *incumbent*, separately from the representations that procedures modify, so procedures can explore more expensive intermediate circuits without losing the best circuit found so far. The state is $s = (E, I, b, h)$, where $E$ stores the symbolic representations, $I$ is the incumbent, $b$ records resource use, and $h$ is the number of decisions remaining. The available actions form a set $A(s)$. Transitions are deterministic: with fixed rewrite rules, enumeration order, and procedure budgets, executing $a \in A(s)$ yields a unique successor $T(s, a)$.

The environment replaces $I$ only after verifying that the candidate circuit is equivalent to $C_0$ and has cost no greater than $c_d(I)$; if the check fails or is unsupported, $I$ is unchanged. Starting from $I_0 = C_0$, every incumbent satisfies

$$I_t \equiv_d C_0, \qquad c_d(I_{t+1}) \preceq_{\text{lex}} c_d(I_t). \tag{1}$$

Correctness of the returned circuit therefore rests on the checker, never on the learned policy; Appendix A states the assumptions and proof.

For learning and search, a scalar utility $u_d(c)$ encodes the cost ordering over the supported bounded range, with larger utility for cheaper circuits. The reward is the improvement in incumbent utility:

$$r(s_t, a_t) = u_d(c_d(I_{t+1})) - u_d(c_d(I_t)). \tag{2}$$

The undiscounted return is the final utility minus the initial utility. An action that only exposes new alternatives receives no immediate reward, but can enable a later cost reduction. Appendix A gives the scalar encoding.

### 3.2 Policy and value model

The policy scores each available procedure using the circuit state and the circuit nodes on which the procedure acts. The observation $o(s)$ consists of a graph of the circuit and its retained alternatives, plus action descriptors and resource-use features; when a Boolean graph exceeds the model's node limit, additional features record how much of the graph the observation retains. A message-passing network embeds the graph nodes and pools their embeddings into a state representation. The action-scoring network combines this representation with the procedure's features and the embeddings of its target nodes. For a quantum sequence-resynthesis action, these targets include the gates in the sequence segment to be replaced. A softmax normalizes the action scores to obtain $\pi_\theta(a \mid o(s))$.

The value model shares the policy's graph encoder and predicts the remaining return $v_\theta(o(s))$ for search. Appendix E specifies the graph encoder and action-scoring network.

### 3.3 Imitation from a finite-horizon teacher

We initialize the policy by imitating a symbolic teacher that enumerates all legal schedules up to a specified horizon $h$ ($h = 2$ for polynomial and $h = 1$ for quantum circuits; Table 5). For each possible first action $a$, the teacher computes the lowest terminal cost reachable after that choice, denoted $Q_h^*(s, a)$. The set $A_h^*(s)$ contains all actions attaining the best of these costs. Retaining the full set allows equally good procedure choices to serve as training targets without an arbitrary tie-break. These action labels are exact for the given action set, horizon, and procedure budgets; Appendix A gives the backward recursion.

The policy is trained to maximize the total probability of the optimal set:

$$\mathcal{L}_{\text{set}}(s) = -\log\left(\sum_{a \in A_h^*(s)} \pi_\theta(a \mid o(s))\right). \tag{3}$$

Any distribution supported on $A_h^*(s)$ minimizes this loss, so the policy may concentrate on one optimal action or spread across several; unlike cross-entropy to a uniform target over ties, the loss prescribes no distribution among them.

### 3.4 SELF-PLAY WITH POLICY-GUIDED SEARCH

We extend training beyond the teacher's short schedules through self-play. At state $s$, the current policy proposes procedures, and the value model estimates the improvement available after each search path. Self-play search uses Gumbel AlphaZero (Danihelka et al., 2022), which samples distinct root actions and allocates simulations by sequential halving, repeatedly discarding the lower-ranked half of the remaining candidates. Search executes the environment's true transition $T(s, a)$ along each edge, with no learned dynamics model, and caches each successor for repeated visits. After search, the environment executes the selected root action, and a new search starts from its successor, repeating until termination.

The policy training target combines the policy scores with estimated returns. Let $\widehat{Q}(s, a)$ denote a search action value; unvisited actions receive the policy-weighted estimate defined in Appendix E.4. Normalizing and scaling these values with $\sigma_s$ gives the target

$$\bar{\pi}(a \mid s) \propto \pi_\theta(a \mid o(s)) \exp\big(\sigma_s(\widehat{Q}(s, a))\big), \qquad a \in A_{\text{exec}}(s). \tag{4}$$

Here $A_{\text{exec}}(s)$ contains actions whose successor states pass the environment's circuit-verification and resource checks. The policy loss is cross-entropy against $\bar{\pi}$. The value target at each state of the executed trajectory is the sum of its subsequent rewards: the improvement in incumbent utility from that state to termination.

The updated network guides the next self-play trajectories. Each training batch mixes these examples with the original teacher labels and *reanalyzed* examples, obtained by searching previously visited states with a newer network. In the quantum configuration, full searches (64 simulations) supply policy and value supervision, while smaller searches (8 simulations) supply value and auxiliary supervision. Appendix E gives the search budgets, losses, and replay sampling.

At deployment, the trained policy acts greedily: it selects the highest-scoring available action at each step until the decision horizon or termination, then returns the incumbent. Neither teacher enumeration nor policy-guided tree search runs during deployment. Deployment time is therefore the time to run the selected procedures, encode graphs, evaluate the network, and check equivalence.

## 4 EXPERIMENTS

The experiments measure quantum circuit costs, compare polynomial optimization schedules, and test which Boolean benchmarks the environment can load and encode. The quantum policy is trained by imitation and self-play; the polynomial policies use imitation alone. The quantum and polynomial inputs are development data: quantum results informed rewrite-rule changes, and polynomial results were inspected during method development. Each learned result comes from a single training run, so variation across seeds is not reported (Agarwal et al., 2021).

### 4.1 QUANTUM OPTIMIZATION AND THE EFFECT OF REWRITE RULES

The quantum comparison evaluates one fixed policy against Quasar on 161 three-qubit circuits. Four families contribute 40 circuits each; the linear-reversible family contributes one after excluding previously used operators. Figure 5 shows the family breakdown, and Appendix D gives the selection protocol. The policy can select at most three macro-actions, and each circuit is limited to 1,800 seconds and 12 GiB.

The baseline is unmodified Quasar with initial and maximum rewrite step 8, a 60-second limit, and integer linear programming (ILP) extraction disabled. Its saved circuits are checked for equivalence to the inputs and compared using the cost tuples in Table 1; all 161 circuits from each optimizer pass the exact operator checks. The two methods run under different limits (1,800 versus 60 seconds), so the comparison is of circuit cost, not speed. Quasar's paper uses a far larger budget (ILP extraction, rewrite steps up to 15, and 7,200 seconds per instance; Appendix D); this comparison covers only the step-8 configuration. During training, outputs of the same step-8 configuration on training circuits also set replay priorities (Appendix E); training and evaluation circuits come from disjoint hash-based splits of circuit operators.

With the final rewrite set, the trained policy obtains a lower-cost circuit on 70 inputs, the same cost on 82, and a higher cost on 9. Of the 70 wins, 2 reduce CX count, 36 reduce total gate count at equal

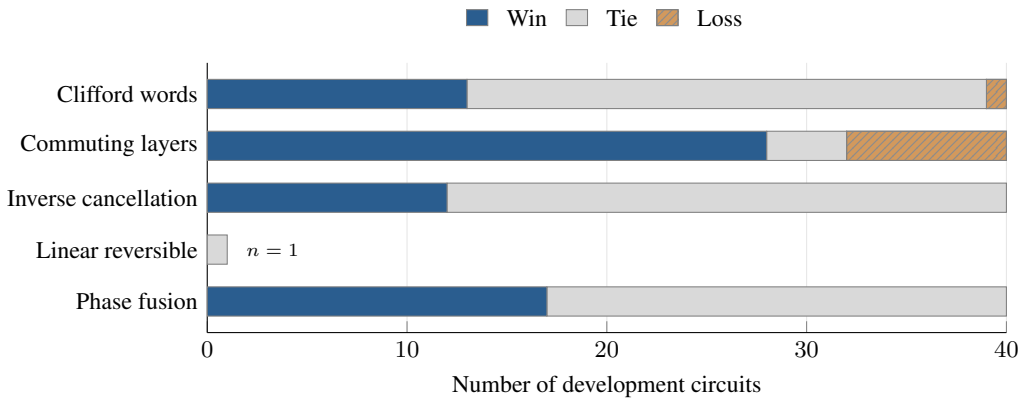


Figure 5: Trained quantum policy with the final rule set against step-8 Quasar on 161 three-qubit development circuits. A win means a lower lexicographic cost $(N_{\mathrm{CX}}, N_{\mathrm{total}}, D, N_{1q})$. Bars show raw counts; the linear-reversible family has one circuit. There are 70 wins, 82 ties, and 9 losses in total.

CX count, and 32 reduce depth at equal counts; the 9 losses split 3/5/1 across the same categories. The policy thus almost always matches Quasar's CX count and wins mainly on secondary costs.

Table 2: Changing the rewrite rules while keeping policy weights and the 161 inputs fixed. Outcomes compare circuit cost with the same step-8 Quasar reference.

| Rewrite set | Wins | Ties | Losses |
|---|---|---|---|
| Initial rewrite set | 46 | 81 | 34 |
| Remove X expansion | 57 | 91 | 13 |
| Add contractions around one SX | 70 | 82 | 9 |

Because the policy selects procedures rather than individual rewrites, the same weights can run with different rule sets (Table 2). The first revision removes the expansion of X into two SX gates. Removing this sound rule cuts losses against Quasar from 34 to 13: under a bounded search, a valid rewrite can hurt. The second revision adds contractions of sequences containing one SX; for example, the identity in Appendix D removes X from an RZ–SX–RZ–X sequence by adjusting the rotation angles. Relative to the middle rule set, it reduces circuit cost on 34 inputs, leaves it unchanged on 124, and increases it on 3.

### 4.2 Polynomial schedule selection

The polynomial comparison evaluates learned procedure selection against fixed two-step schedules. Training uses 800 circuits and 6,958 teacher-labeled decisions; evaluation uses 400 separate development circuits over $\mathbb{F}_2$, $\mathbb{F}_3$, and $\mathbb{F}_5$, spanning five families and four difficulty levels (Appendix B). The comparison includes two imitation-only policies: an optimal-set policy trained with Equation (3) alone, and a cost-sensitive policy whose loss adds the expected excess scalar-free operation count, using the teacher's per-action circuit costs (Appendix E). The cost-sensitive policy is trained from scratch for 1,024 full-batch gradient steps.

The three fixed schedules use the same symbolic procedures available to the policies (Table 3). The decomposition procedure searches ways of splitting polynomial terms into smaller subproblems; shared-graph saturation rewrites the resulting instruction graph. The factorization portfolio tries several decompositions, while Horner rewriting arranges a polynomial into nested multiply-add expressions. Each method takes at most two macro-actions. Every circuit returned by the five methods passes the exact coefficient check against its input.

The cost-sensitive policy's 4,763 scalar-free operations are fewer than those of every fixed schedule (12 fewer, or 0.25%, than the best, portfolio followed by Horner rewriting) and 15 fewer than those

Table 3: Polynomial cost totals on the same 400 development circuits. $S$ is scalar-free operation count and $\sum D$ sums circuit depths. The final row takes the lowest scalar-free operation count per input after running all three fixed schedules.

| Policy or schedule | $S$ | $N_\times$ | $N_+$ | $\sum D$ | $N_{\text{total}}$ |
|---|---|---|---|---|---|
| Decomposition → shared-graph saturation | 5,296 | 4,230 | 1,544 | 2,468 | 5,774 |
| Portfolio → portfolio | 4,807 | 3,907 | 1,391 | **2,429** | 5,298 |
| Portfolio → Horner | 4,775 | 3,853 | 1,393 | 2,438 | 5,246 |
| Optimal-set imitation | 4,778 | 3,851 | 1,391 | 2,438 | 5,242 |
| Cost-sensitive imitation | **4,763** | **3,830** | **1,387** | 2,443 | **5,217** |
| Best of three fixed schedules | 4,710 | scalar-free count only | | | |

of the optimal-set policy. Relative to the Horner schedule, it reduces operation count on 28 inputs, ties on 348, and increases it on 24; its summed circuit depth is five greater.

Running all three fixed schedules and selecting their cheapest output separately for each input gives 4,710 operations, 53 (1.1%) fewer than the policy. The policy nevertheless improves on all three fixed schedules on 10 inputs because it can choose procedure sequences outside those schedules. Adding a fourth fixed schedule, chosen using the training set, lowers the portfolio total to 4,684 operations. Appendix F compares that portfolio with one containing a refitted policy and reports the additional supervised fits.

A further 256 supervised updates on the original training set give a policy with 0.27% fewer scalar-free operations than portfolio followed by Horner rewriting. Summed over two repetitions, this policy uses 1.57 times the CPU time of that schedule. This measurement includes symbolic execution and equivalence checking, not just network evaluation. Appendix F specifies the timed phases and reports both repetitions.

### 4.3 Boolean benchmark encoding

The Boolean study tests whether the environment can load and encode all 100 public IWLS 2026 benchmarks (IWLS 2026 Programming Contest Organizers, 2026), each given as its published minimum-area reference circuit. A case succeeds if, within one attempt limited to 25 CPU seconds, 30 wall seconds, and 4 GiB, the environment initializes and verifies the circuit state, enumerates actions, and produces a model input that round-trips through serialization without loss (Appendix C). The environment encodes 98 cases: 75 with the complete graph and 23 whose model inputs omit some graph context under the 8,192-node limit. One failure exceeds the 20,000-node source-circuit limit; the other reaches the CPU limit during model encoding. The study runs no policy and measures no reduction in Boolean circuit cost.

### 4.4 Limitations

The experiments do not isolate the benefit of self-play after imitation. A controlled comparison would start imitation-only and self-play continuations from the same checkpoint, keep the symbolic rules fixed, and match training and evaluation budgets on held-out circuits. We also lack a matched timing comparison: Quasar's circuits were reused from earlier runs, so its runtime was not remeasured, and the timed polynomial policies are slower than the fixed schedule they beat. Whether learned policies reduce inference-time search therefore remains open.

## 5 Related work

**Equality saturation and search control.** The `egg` library provides reusable e-graph infrastructure (Willsey et al., 2021), and Quasar combines graph and sequence representations for quantum circuit optimization (Yang et al., 2026b). MCTS-GEB uses Monte Carlo tree search to guide e-graph construction under resource limits (He et al., 2023). Unlike MCTS-GEB, which runs tree search for each input, we use symbolic search to train a procedure-selection policy that is deployed without tree search.

**Learned circuit optimization.** Learned optimizers differ in the decisions assigned to the policy. Quarl, built on the verified transformations generated by Quartz (Xu et al., 2022), learns which gate location and local transformation to choose (Li et al., 2024). At the level of optimization procedures, DRiLLS learns Boolean synthesis flows (Hosny et al., 2020), and ABC-RL guides tree search over synthesis recipes with a pretrained policy, using retrieval to temper that policy on unfamiliar circuits (Chowdhury et al., 2024). CircuitPilot also learns procedure selection: one action can run several rewrite rounds or invoke resynthesis. Unlike these methods, we supervise these decisions with exact optimal first-action sets from a finite-horizon teacher before training on action distributions from policy-guided search.

**Polynomial circuit construction.** CircuitBuilder treats arithmetic circuit construction as a single-player reinforcement learning problem, and FactorLibrary decomposes polynomial targets into recursive subgoals (Zhang et al., 2026; Pandey et al., 2026). We instead start from a given circuit; algebraic search runs inside the factorization-portfolio action, and the policy learns when to invoke it relative to other procedures.

**Imitation and self-play.** Supervised initialization followed by self-play underlies AlphaGo (Silver et al., 2016); AlphaTensor combines synthetic demonstrations with search-based reinforcement learning for algorithm discovery (Fawzi et al., 2022). We apply this training sequence to circuit-optimization procedures, with policy targets from Gumbel AlphaZero (Danihelka et al., 2022). Our demonstrations differ: they are exact optimal first-action sets from exhaustive search, not human games (AlphaGo) or synthetic decompositions (AlphaTensor).

## 6 Conclusion

We introduced CircuitPilot, which learns reusable circuit-optimization policies from symbolic search through imitation learning followed by single-player self-play. The same formulation applies to polynomial, Boolean, and quantum circuits, with domain-specific equivalence checks. On development data, the quantum policy matches or beats step-8 Quasar on 152 of 161 circuits, with results that depend strongly on the rewrite rules, and the imitation-only polynomial policy slightly outperforms each fixed schedule; the Boolean study tests which benchmark circuits can be encoded as model inputs. Establishing the benefit of self-play and any reduction in optimization runtime requires controlled training comparisons and matched runtime measurements.

### AI use statement

Generative AI tools, including OpenAI Codex, were used for literature discovery, manuscript drafting and revision, figure construction, research planning, implementation and debugging, synthetic task-generation code, experiment orchestration, analysis of saved results, and development of the arguments in Appendix A. Reported numerical results come from recorded experiments; reporting scripts check their aggregation. The equivalence of reported circuits is established by the domain-specific checkers described in the paper, not by AI-generated reasoning. The authors are responsible for the final claims, code, and disclosure.

### Reproducibility statement

Sections 2–4 specify the circuit environments, learning procedure, and comparison scope. Appendices B–E give domain details, reference settings, and training configurations. Appendix G describes the data and scripts for checking reported aggregates and distinguishes these checks from the additional artifacts needed to reproduce training. Code is available in an anonymous repository at https://anonymous.4open.science/r/circuitpilot-8F0E/.

## References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C. Courville, and Marc G. Bellemare. Deep reinforcement learning at the edge of the statistical precipice. In *Advances in Neural Information Processing Systems*, volume 34, pp. 29304–29320, 2021. URL https://proceedings.neurips.cc/paper/2021/hash/f514cec81cb148559cf475e7426eed5e-Abstract.html.

Animesh Basak Chowdhury, Marco Romanelli, Benjamin Tan, Ramesh Karri, and Siddharth Garg. Retrieval-guided reinforcement learning for Boolean circuit minimization. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=0t1O8ziRZp.

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

Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli. Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature*, 610:47–53, 2022. doi: 10.1038/s41586-022-05172-4. URL https://doi.org/10.1038/s41586-022-05172-4.

Guoliang He, Zak Singh, and Eiko Yoneki. MCTS-GEB: Monte Carlo tree search is a good e-graph builder. In *Proceedings of the 3rd Workshop on Machine Learning and Systems*, 2023. URL https://arxiv.org/abs/2303.04651.

Abdelrahman Hosny, Soheil Hashemi, Mohamed Shalan, and Sherief Reda. DRiLLS: Deep reinforcement learning for logic synthesis. In *25th Asia and South Pacific Design Automation Conference*, 2020. URL https://arxiv.org/abs/1911.04021.

IWLS 2026 Programming Contest Organizers. IWLS 2026 logic synthesis contest: Benchmarks and reference circuits, 2026. URL https://github.com/alanminko/iwls2026-ls-contest.

Zikun Li, Jinjun Peng, Yixuan Mei, Sina Lin, Yi Wu, Oded Padon, and Zhihao Jia. Quarl: A learning-based quantum circuit optimizer. *Proceedings of the ACM on Programming Languages*, 8(OOPSLA1):555–582, 2024. doi: 10.1145/3649831. URL https://doi.org/10.1145/3649831.

Rohan Pandey, Michael Ruofan Zeng, Weikun K. Zhang, Kaijie Jin, Naomi Morato, Archit Ganapule, Bhaumik Mehta, and Jarod Alper. FactorLibrary: From polynomials to circuits via recursive subgoals, 2026. URL https://arxiv.org/abs/2606.25394. 3rd AI for Math Workshop, ICML.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529:484–489, 2016. doi: 10.1038/nature16961. URL https://doi.org/10.1038/nature16961.

Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. egg: Fast and extensible equality saturation. *Proceedings of the ACM on Programming Languages*, 5(POPL), 2021. doi: 10.1145/3434304. URL https://doi.org/10.1145/3434304.

Mingkuan Xu, Zikun Li, Oded Padon, Sina Lin, Jessica Pointing, Auguste Hirth, Henry Ma, Jens Palsberg, Alex Aiken, Umut A. Acar, and Zhihao Jia. Quartz: Superoptimization of quantum circuits. In *Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, 2022. doi: 10.1145/3519939.3523433. URL https://doi.org/10.1145/3519939.3523433.

Ganxiang Yang, Paige Raun, Runzhou Tao, and Ronghui Gu. Equality saturation for quantum circuit optimization: Artifact, 2026a. URL https://zenodo.org/records/19571754. Version 3.

Ganxiang Yang, Paige Raun, Runzhou Tao, and Ronghui Gu. Equality saturation for quantum circuit optimization. *Proceedings of the ACM on Programming Languages*, 10(PLDI):174–199, 2026b. doi: 10.1145/3808254. URL https://doi.org/10.1145/3808254.

Weikun K. Zhang, Rohan Pandey, Bhaumik Mehta, Kaijie Jin, Naomi Morato, Archit Ganapule, Michael Ruofan Zeng, and Jarod Alper. CircuitBuilder: From polynomials to circuits via reinforcement learning, 2026. URL https://arxiv.org/abs/2603.17075. ICLR Workshop on AI with Recursive Self-Improvement.

# A CIRCUIT EQUIVALENCE, COST GUARANTEES, AND BOUNDED SUPERVISION

**Assumptions.** The initial incumbent is the supplied circuit. A candidate circuit replaces it only if a sound domain-specific checker verifies that the candidate is equivalent to the input circuit or to a predecessor already verified as equivalent to that input. Cost is recomputed from the candidate circuit, and replacement requires a circuit-cost tuple no greater in lexicographic order. The quantum environment breaks equal-cost ties by artifact hash. A failed or unsupported check leaves the incumbent unchanged. The argument below assumes the checker, parser, and circuit-cost computation satisfy these specifications; it is not a machine-checked proof of the entire implementation.

**Incumbent preservation.** The initial circuit satisfies (1). If a transition retains the incumbent, the invariant is unchanged. If it accepts a candidate circuit, transitivity of circuit equivalence makes the new incumbent equivalent to $C_0$, and the cost comparison gives a lexicographically nonincreasing cost. By induction, every incumbent along every finite trajectory is equivalent to $C_0$, and cost never increases, whatever the policy chooses.

**Optimal first actions within a finite horizon.** Let $V_h^*(s)$ be the lowest incumbent cost reachable within $h$ decisions. The teacher evaluates

$$\begin{aligned} V_0^*(s) &= c_d(I(s)), & Q_h^*(s,a) &= V_{h-1}^*(T(s,a)), \\ V_h^*(s) &= \min_{\text{lex},\, a \in A(s)} Q_h^*(s,a), & A_h^*(s) &= \{a \in A(s) : Q_h^*(s,a) = V_h^*(s)\}. \end{aligned} \tag{5}$$

For a terminal state $s$, $V_h^*(s) = c_d(I(s))$ for every $h$. The recurrence is exact by induction on $h$: after every legal first action has been evaluated, its successor value gives the best cost within the remaining $h-1$ decisions. Taking the minimum gives $V_h^*$ and retaining every minimizer gives $A_h^*$. Resource caps inside a procedure are deterministic, so they are part of $T$ and do not affect exactness. An interrupted teacher that omits a legal branch, however, cannot certify the complete set of optimal first actions.

This guarantee concerns the given action set and horizon; like exact extraction (Section 2), it does not establish a minimum over all circuits computing the same function.

**Scalar utility.** For a circuit-cost tuple $c = (c_1, \ldots, c_m)$ with nonnegative integer components bounded by $B$, the lexicographic order can be encoded as

$$\rho(c) = \sum_{j=1}^{m} c_j (B+1)^{m-j}, \qquad u(c) = -\frac{\rho(c)}{(B+1)^{m-1}}.$$

Thus a lower cost has a larger utility. The implementation checks the representable range before converting circuit costs to floating-point utility values; the quantum configuration uses $B = 1023$ for four components. Unsupported ranges are rejected. Exact cost vectors, not neural utility predictions, determine reported wins and incumbent replacements.

# B POLYNOMIAL IMPLEMENTATION AND DATA

## B.1 POLYNOMIAL OPERATIONS, CIRCUIT EQUIVALENCE, AND SHARING

Table 4 describes the twelve available actions. The teacher and policy use the same implementations and resource limits. In particular, the decomposition and portfolio actions perform symbolic search internally; one policy decision need not be one algebraic rewrite.

Polynomial circuits are stored as graphs of shared instructions in static single assignment (SSA) form. Each instruction defines one intermediate value, which can be used by several later instructions or circuit outputs.

Each shared instruction reachable from an output (an *active* instruction) is charged once. An addition of $k$ operands contributes $k-1$ additions. A multiplication with $k$ operands contributes $k-1$ multiplications to the total multiplication count; if only $r$ operands are nonconstant polynomials,

Table 4: Polynomial actions. Search and rewriting are subject to the state's remaining resource budget.

| Action | Effect |
|---|---|
| Expression rewriting | Apply one rewrite layer to the expression e-graph. |
| Shared-graph rewriting | Apply one rewrite layer to the graph of shared instructions. |
| Sharing | Merge expression alternatives into the graph of shared instructions. |
| Inlining | Merge shared-graph alternatives into the expression e-graph. |
| Local factorization | Apply one layer of distributive factoring. |
| Horner rewriting | Add nested multiply–add forms for the outputs. |
| Expansion | Add expanded polynomial expressions for the outputs. |
| Bounded circuit synthesis | Enumerate shared circuits up to a gate bound. |
| Term decomposition | Search additive splits and factorizations into shared instructions. |
| Factorization portfolio | Retain several bounded decompositions for joint extraction. |
| Extraction | Select a circuit from the current alternatives. |
| Stop | Return the incumbent. |

it contributes $\max(0, r-1)$ to the scalar-free operation count. Constants and variables are free leaves. Depth is computed over active dependencies with binary associative evaluation of multi-input operations. Table 3 reports the sum of these per-input depths.

The checker normalizes polynomials using the given coefficient field and variable ordering, then compares the coefficient representations of corresponding circuit outputs.

### B.2 Training distribution and baselines

The data generator samples circuits from five families: common-factor expressions, Horner chains, random directed acyclic graphs, shared multi-output circuits, and elementary symmetric polynomials. Each family has four difficulty levels, with 40 training and 20 development circuits generated per combination. Circuits computing identical output polynomials cannot appear in both sets. Results on the development set were inspected during method development, so it is not a held-out test set. For fields $p = 2, 3, 5$, the training counts are $(251, 267, 282)$ and the development counts are $(124, 140, 136)$.

The three fixed schedules are term decomposition followed by shared-graph rewriting, two calls to the factorization portfolio, and the portfolio followed by Horner rewriting. For scalar-free operation counts $S_{ik}$ on input $i$ under schedule $k$, selecting the cheapest result for each input gives $\sum_i \min_k S_{ik} = 4710$. Computing this total requires all three schedules to be run. The policy's excess is $(4763 - 4710)/4710 \approx 1.1\%$.

## C Boolean circuit extraction and benchmark input encoding

### C.1 Semantic classes and circuit extraction

An $n$-input truth table is represented by a $2^n$-bit integer $t$, with complement mask $J = 2^{2^n} - 1$. A semantic class stores $\kappa(t) = \min(t, t \oplus J)$; an edge $(\kappa, \epsilon)$ with polarity $\epsilon \in \{0, 1\}$ denotes $\kappa \oplus \epsilon J$. The checker evaluates each candidate gate, called a producer, as a truth table and verifies that it belongs to its semantic class. Conversions between expression and network representations require matching truth tables at corresponding circuit outputs. These tables require space exponential in the number of circuit inputs, even for a small circuit graph.

For circuit extraction, binary variables $y_c$ indicate whether class $c$ is used in the selected circuit, and $x_{c,i}$ indicates whether producer $i$ is selected for a nonleaf class $c$. Leaves are circuit inputs and constants. Output roots must be active, an active nonleaf class selects one producer, and that producer activates its required fanin classes:

$$y_r = 1 \quad (r \text{ an output root}), \qquad \sum_i x_{c,i} = y_c, \qquad x_{c,i} \le y_d \quad (d \text{ a required fanin class}).$$

Every active nonroot class must be used by a selected gate, and integer ordering variables require each selected gate to follow its inputs. These constraints exclude disconnected selections and cycles.

Leaves and edge complements contribute no gate cost, and a shared selected producer is charged once. The OR-Tools CP-SAT solver runs with one worker to minimize successive circuit-cost components, keeping earlier optima fixed. Only an optimal status at every stage certifies a minimum-cost circuit among those represented in the graph. Otherwise, the environment may retain a feasible circuit after a separate check of its equivalence to the input, without certifying minimum cost.

Small-instance exhaustive enumeration and canonicalization up to input negation, input permutation, and output negation (NPN) are limited to four inputs. The same boundary-input limit applies to cut resynthesis within a larger circuit. The four-input limit applies to these local subproblems, not to the whole circuit.

### C.2 Large-graph encoding protocol

The input adapter retains the supplied circuit graph directly instead of unfolding shared computations into expression trees. The input-encoding study uses circuits `ex200`–`ex299` from IWLS 2026, which provide completely specified multi-output truth tables and published area–delay reference frontiers (IWLS 2026 Programming Contest Organizers, 2026). Each benchmark uses one minimum-area reference circuit, with area ties broken by delay and then filename. Each case must pass input validation, circuit-state initialization and verification, snapshot construction, action enumeration, model encoding, serialization and deserialization of that encoding, and a final check that its source data are unchanged. The test evaluates no policy and executes no optimization action.

Each case has one attempt, with limits of 25 CPU seconds, 30 wall seconds, and 4 GiB of address space. Model observations retain at most 8,192 nodes; source circuit graphs allow at most 20,000. A truncated model input preserves the action-target nodes but may omit other graph context; it does not alter the circuit whose output truth tables are compared with the benchmark specification. The two failures are `ex298`, whose reference circuit has 21,553 nodes, and `ex299`, which exceeds the CPU limit during model encoding.

## D Quantum circuit representations and Quasar settings

### D.1 Circuit-equivalence checks and sequence extraction

For rotation angles that are rational multiples of $\pi$, the checker represents matrix entries in a cyclotomic field, a number field generated by a root of unity. Dividing by the first nonzero operator entry gives a canonical representative up to global phase. The whole-operator checker is bounded to three qubits, phase order 256, and cyclotomic degree 128. For circuits with more than three qubits, the checker requires a supported derivation from the input circuit: local operator identities, permutations preserving per-wire dependencies, and checked representation conversions. Local identities compose under ordinary sequential and tensor composition. The checker does not automatically extend an identity that holds only up to global phase to a new controlled operation, because a global phase becomes a relative phase under control.

A node $(P, g)$ in the sequence e-graph appends gate $g$ to a prefix class $P$. The e-graph merges constructions with equivalent prefixes and the same appended gate, avoiding enumeration of every full gate sequence. Extraction first minimizes $(N_{\mathrm{CX}}, N_{\mathrm{total}})$, then retains the Pareto frontier of per-wire ready times among derivations tied on those two counts. A wire's ready time is the depth after its last gate; one vector dominates another if it is no larger on any wire and smaller on at least one. Keeping all nondominated vectors matters because a prefix that is deeper on some wires can give a shallower circuit once later gates are appended. In the gate-graph representation, a separate budgeted scan of complete output circuits certifies a minimum-cost circuit in that graph only when every represented output circuit has been considered. Gate-graph extraction uses each selected quantum wire value once and retains every output of a selected multi-qubit gate; it cannot introduce arbitrary fanout as in a classical circuit (no-cloning).

The final rule set retains the two CX expansions, each of which rewrites a pair of CX gates as three, but omits the rule $\mathrm{X} \to \mathrm{SX}; \mathrm{SX}$ and includes contractions of four-gate sequences containing one SX. One identity in this family, written in gate execution order and interpreted modulo global phase, is

$$R_Z(a);\ \mathrm{SX};\ R_Z(b);\ \mathrm{X} \ \equiv\ R_Z(a+\pi);\ \mathrm{SX};\ R_Z(\pi-b).$$

The contraction is enabled only when the number of non-identity gates decreases, since zero-angle rotations can make syntactic length differ from gate cost. The rewrite–extract procedure has budgets of eight rewrite layers, 200,000 match probes, and 50,000 e-nodes; reaching a budget terminates exploration without requiring equality saturation to reach a fixed point.

### D.2 Baseline and development reuse

The baseline uses Quasar commit `89dfae50fe3507fa9f212e6780f5d52443afdd6c`. The initial and maximum step are fixed to eight, ILP is disabled, and the time limit is 60 seconds. Quasar's released paper artifact instead describes ILP-enabled runs, instance-specific starting steps, escalation up to step 15, and a 7,200-second per-instance budget (Yang et al., 2026a). The reported comparison applies only to this step-8 Quasar configuration.

Case selection excluded circuits whose operators had appeared in two earlier development sets, retaining 40 new circuits from each of four families. The fixed linear-reversible development set contained 17 distinct operators, of which 16 had already been used, leaving one for this comparison. Outcomes on these 161 circuits then informed the final rule revision, so the final comparison uses development data rather than an untouched test set.

All three rule sets use the same policy weights. The initial and final rule sets were rerun; saved circuits for the middle rule set and for Quasar were re-verified. All 161 cases complete, with no failed case scored as a tie. Reusing saved Quasar circuits provides no new measurement of Quasar's runtime.

## E Model and training settings

### E.1 Message passing and action scoring

Each initial node embedding is the sum of a numeric projection and embeddings of its kind, type, and representation view. Let $h_v$ be the embedding of node $v$ and $e_{uv}$ the embedding of the relation on edge $u \to v$. A message-passing layer computes

$$m_v = \mathrm{mean}_{u \to v}(W_s h_u + W_t h_v + W_r e_{uv} + b),$$
$$h'_v = \mathrm{LayerNorm}\left(h_v + \mathrm{Dropout}(\mathrm{GRU}(m_v, h_v))\right).$$

Each layer has separate parameters. Nodes without incoming edges receive a zero aggregate; Table 5 gives the dropout rates.

A learned sigmoid gate pools node embeddings into a normalized weighted mean. The encoder concatenates this vector with the sum of projected state-summary features and projected features describing the retained graph context, then projects the result to obtain a state context $g$. An action embedding $z_a$ is the sum of embeddings of its numeric, kind, family, and representation features. A target context $t_a$ averages transformed embeddings of the action's target nodes, roles, and positions. The action-scoring network projects $[g, z_a, t_a, g \odot z_a]$ to the hidden width and scores it with a two-layer MLP with GELU activation. During greedy inference, the policy breaks exact logit ties in canonical action order.

### E.2 Polynomial fit

Let $q_1(s, a)$ denote the first component of the teacher's cost $Q^*_h(s, a)$. The cost-sensitive objective combines optimal-set likelihood with the expected excess scalar-free operation count:

$$\mathcal{L}_{\mathrm{poly}}(s) = \mathcal{L}_{\mathrm{set}}(s) + \sum_{a \in A(s)} \pi_\theta(a \mid o(s)) \left[ q_1(s, a) - \min_{a' \in A(s)} q_1(s, a') \right]. \tag{6}$$

The first term uses all first actions attaining the lowest lexicographic circuit cost within the teacher's horizon; the second uses only the scalar-free operation count.

The cost-sensitive model is trained from scratch with seed 20261003. It uses 16 numeric features per node, 24 per action, and 96 for the state summary. Each of the 800 input circuits has weight

Table 5: Training configurations for the polynomial imitation policy and the final quantum policy.

| Setting | Polynomial | Quantum |
|---|---|---|
| Hidden width / message layers | 128 / 6 | 256 / 6 |
| Dropout | 0 | 0.05 |
| Teacher macro horizon | 2 | 1 (includes rewrite–extract) |
| Deployment macro horizon | 2 | 3 |
| Teacher data | 800 circuits; 6,958 decisions | 2,048 regenerated examples |
| Reported endpoint | 1,024 supervised updates | policy at step 14,621 |
| Weight decay | $10^{-5}$ | $10^{-4}$ |
| Gradient clipping norm | 1 | 8 |
| Search macro horizon | — | 3 |
| Full / small search simulations | — | 64 / 8 |
| Candidate root actions | — | at most 16 |

$1/800$, divided equally among its teacher-labeled decisions, so inputs with more reachable decisions do not receive greater weight. Both terms in (6) have coefficient one; the excess operation count is neither normalized nor clipped.

AdamW updates all model parameters with the cosine learning-rate schedule

$$\eta_t = 10^{-5} + \tfrac{1}{2}(10^{-3} - 10^{-5}) \left[1 + \cos\left(\pi \frac{t-1}{1023}\right)\right], \quad 1 \le t \le 1024.$$

Eight single-threaded gradient replicas process batches of graphs and reduce a full-data gradient at each update. Evaluation uses the final checkpoint; no checkpoint was selected on development scores. Training took 19,413 seconds (5.4 hours) of wall-clock time, including final checks and serialization but excluding teacher-label generation.

### E.3 Quantum continuation

The final quantum policy continues an earlier model trained from scratch for 12,000 imitation updates, followed by 47 updates after the switch to self-play. After the rewrite–extract procedure was added to the teacher's action set, training restarted from that model and its exponential-moving-average (EMA) weights, without the old replay or optimizer state. The teacher regenerated 2,048 labeled examples for 1,000 supervised updates before the switch to self-play. The learning rate is $5 \times 10^{-5}$ with 128 warmup steps, EMA decay is 0.99, and batches contain 32 examples with eight gradient-accumulation steps.

Each new self-play trajectory receives 64 simulations per decision with probability 0.25, and eight otherwise. The 64-simulation searches consider at most 16 candidate root actions and supply policy, value, and auxiliary targets; the eight-simulation searches supply value and auxiliary targets. After the supervised warm start, training batches contain 10% teacher-labeled examples, 80% self-play examples, and 10% reanalyzed examples. The rule-set comparison uses the same final policy throughout, without further training; this continuation supplies no imitation-only control.

### E.4 Search targets

At a search root $s$, let $p_a = \pi_\theta(a \mid o(s))$, let $N_a$ be the visit count of action $a$, and let $q_a$ be its mean backed-up return when $N_a > 0$. Let $V = \{a : N_a > 0\}$ and $N = \sum_a N_a$. Gumbel AlphaZero assigns unvisited actions a mixture of the network value and the policy-weighted returns of visited actions:

$$q_{\text{mix}} = \frac{v_\theta(o(s)) + N \sum_{a \in V} \frac{p_a}{\sum_{b \in V} p_b} q_a}{1 + N}, \qquad \widehat{Q}(s,a) = \begin{cases} q_a, & N_a > 0, \\ q_{\text{mix}}, & N_a = 0. \end{cases}$$

Visited-action probabilities are floored at the smallest positive normal floating-point value before normalization in this weighted mean. For $\Delta = \max_a \widehat{Q}(s,a) - \min_a \widehat{Q}(s,a)$, the quantum config-

uration rescales action values as

$$\sigma_s(q) = \begin{cases} 0.1(50 + \max_a N_a)\dfrac{q - \min_a \widehat{Q}(s,a)}{\Delta}, & \Delta > 10^{-8}, \\ 0, & \Delta \le 10^{-8}. \end{cases}$$

Adding these scaled values to the policy logits and applying a softmax gives the distribution in Equation (4) before action masking. To identify incomplete transitions, the environment executes every root action once, reusing cached transitions. A transition stopped by a resource limit can remain executable if it yields a verified circuit under the environment's acceptance rules or a complete successor for the given bounded procedure. Other incomplete transitions receive zero target probability, and the remaining probabilities are renormalized. If their sum is numerically zero, the target assigns all probability to the executed root action: the search's selection if it is executable, and otherwise the executable action with the highest completed value.

For an executed trajectory ending at step $T$, the value target is $G_t = \sum_{k=t}^{T-1} r(s_k, a_k) = u_d(c_d(I_T)) - u_d(c_d(I_t))$. Full searches supply both the action distribution and this return target; smaller searches supply return and auxiliary targets. The greedy comparison described below affects only replay priority.

### E.5 Quantum losses and replay priorities

Quantum training fits policy and return targets together with auxiliary predictions of action values, terminal costs, and the observed outcomes of symbolic execution (Table 6). Return and cost predictions use the pooled state representation; action-specific predictions use the action context. Missing targets are masked out. Losses are averaged with replay importance weights, and action-level losses are balanced across states. Prioritized replay uses exponent 0.6 and importance-sampling exponent 0.4.

Table 6: Loss terms in the quantum continuation. The policy loss is Equation (3) for teacher-labeled examples and cross-entropy for search targets. Each remaining row describes a separate prediction or ranking loss.

| Target | Loss | Weight |
|---|---|---|
| Policy | Optimal-set loss or cross-entropy | 1 |
| Remaining return | Smooth L1 | 1 |
| Action value | Smooth L1 | 0.5 |
| Action ranking | Cross-entropy, temperature 0.1 | 0.10 |
| Terminal cost vector | Mean Smooth L1 across components | 0.25 |
| $\log(1 + \text{e-graph nodes added})/8$ | Smooth L1 | 0.10 |
| $\log(1 + \text{extractor calls})$ | Smooth L1 | 0.05 |
| Presence of a proof record | Binary cross-entropy | 0.05 |
| Observed resource cutoff | Binary cross-entropy | 0.05 |

The ranking term matches softmax distributions over available action-value targets and predictions, both at temperature 0.1. This term applies only when at least two observed action values differ. Resource-cutoff labels mark a limit reached during execution.

Replay priority depends on a comparison between the search trajectory and a greedy rollout from the same state. Greedy execution takes the highest-scoring policy action at each step. Both final circuits are checked for equivalence to the input circuit, and the trajectories are ordered lexicographically by

$$\big(c_d(I_{\text{terminal}}), w_{\text{suffix}}\big), \qquad w = (W, N_{\text{added}}, H). \tag{7}$$

These counters describe execution of the remaining macro-action sequence from the state being evaluated, not the tree search that generated it. Aggregate work $W$ sums rewrite layers, enumerated regions, synthesis effort, extractor calls, rewrite-match counts, and domain-specific work units. $N_{\text{added}}$ counts added e-graph nodes; $H$ counts macro-action invocations. Attempted transitions contribute to these counters even if resource limits reject them. Cost takes precedence over all work counters. For examples with accepted Quasar reference outputs, strict improvements over greedy execution receive a replay-priority floor of 12. Other examples receive floors of 6, 3, or 2 when the search output loses, ties, or wins against Quasar, respectively. Full-search policy targets are retained in all these cases.

# F Additional polynomial comparisons

## F.1 Per-input polynomial comparisons

Relative to the optimal-set policy, the cost-sensitive policy produces lower scalar-free operation counts on 26 development inputs, equal counts on 352, and higher counts on 22. Relative to the minimum count from the three fixed schedules on each input, its counts are lower on 10 inputs, equal on 343, and higher on 47.

## F.2 Additional supervised fitting

These fits update only the final two-layer policy scorer, holding the graph encoder and feature projections fixed. Starting from the cost-sensitive policy, 256 updates on the original 800 training circuits give 4,762 scalar-free operations on the same development set; 256 updates on 4,000 training circuits give 4,760. A separate 4,096-update fit on 4,000 circuits gives 4,786 despite a lower cost on its training circuits. All three fits start from the same checkpoint.

## F.3 CPU time for the optimization pipeline

The timing study measures optimization CPU time for the 256-update policy, the cost-sensitive policy it started from, and two fixed schedules, each run twice on the 400 development inputs with every returned circuit checked.

Table 7: CPU time for the polynomial optimization pipeline, summed over 400 development inputs for each of two repetitions. Output circuits and costs are unchanged between repetitions.

| Method | Total $S$ | CPU seconds, repeat 1 | CPU seconds, repeat 2 |
|---|---|---|---|
| 256-update policy | 4,762 | 352.010 | 402.018 |
| Initial cost-sensitive policy | 4,763 | 385.858 | 439.768 |
| Portfolio $\rightarrow$ portfolio | 4,807 | 347.415 | 396.388 |
| Portfolio $\rightarrow$ Horner | 4,775 | 225.104 | 255.087 |

The timed phases include setup, symbolic optimization with equivalence checks, artifact writes, and additional verification of SSA circuit representations; they exclude module imports and result auditing. Measurements were collected on a host shared with other jobs. The learned policies trade additional CPU time for fewer operations than portfolio followed by Horner rewriting.

## F.4 Four-schedule portfolios

A fourth fixed schedule, Horner rewriting followed by expression saturation, was selected using only the original training set. Taking the lowest-cost circuit from these four schedules gives 4,684 scalar-free operations on the 400 development inputs. A portfolio of the original three schedules and a separately refitted policy (trained on 4,000 circuits; 4,765 operations on its own) gives 4,702, so the all-fixed portfolio is 18 operations better. Both portfolios execute four optimization schedules per input.

# G Reporting data and reproducibility

Code is available at https://anonymous.4open.science/r/circuitpilot-8F0E/. The reporting material includes per-circuit costs, saved-artifact hashes, and vector figure sources. The reporting script recomputes the main tables and quantum plot from the data extract and, when the original runs are available, checks the pinned result hashes and stored quantum circuit bytes. These checks do not rerun circuit-equivalence verification.

Reproducing training requires frozen source code, dependencies, checkpoints, inputs, and equivalence-check records beyond this extract. Complete training-compute accounting and independent training runs are also unavailable for the reported comparisons.