TRAINING BEYOND REACH: RECURSIVE SELF-IMPROVEMENT WITH VERIFIABLE CIRCUIT TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Circuit-design agents must coordinate structural edits, repair, and tuning to satisfy coupled specifications, yet complete-design successes can be scarce, limiting self-training. We propose VERA (Verifiable Escalation and Reduction for self-training Agents), which uses verified circuits to supply recovery practice and seed further task generation. Reduction changes the starting state to lower the remaining decision burden, normally retaining the seed's full specifications, and certifies a feasible recovery path. Within successful learner rollouts, action selection identifies verified progress, repair, and completion steps for supervision. After each update, the improved learner searches for stronger-specification seeds that yield a fresh recovery-task bank under a fixed verifier. A bootstrap bank derived from human-constructed seeds supplies 1,200 selected actions. One update raises the 9B learner's held-out complete-design success from 0/180 to 77/180 (42.8%). Success-trajectory SFT on the same trajectories reaches 47/180 (26.1%); VERA also improves performance on two external circuit benchmarks. After three updates, the learner reaches 123/180 (68.3%), exceeding fixed-bank recollection and frozen-search controls. These results support linking accessible recovery practice to model-driven renewal of circuit training tasks. Our source code is at `https://anonymous.4open.science/r/anonymous_code1-F4C3/`.

1 INTRODUCTION

Circuit design couples topology construction and parameter tuning under gain, bandwidth, power, and other requirements (Lai et al., 2025; 2026), making it a representative long-horizon engineering task. Simulation Program with Integrated Circuit Emphasis (SPICE) provides executable feedback and acceptance checks from which agents can learn.

Self-training on accepted model outputs (Zelikman et al., 2022; Singh et al., 2024) needs attainable, informative experience. Complete-design tasks near or beyond the learner's capabilities can yield few successes under a fixed budget: our base learner fails every held-out complete-design attempt. Early netlist or functional errors can block later tuning even when local decisions remain within reach (Figure 1). Successful rollouts can still mix useful and ineffective edits, so endpoint acceptance alone cannot identify useful intermediate actions. Curriculum learning and adaptive task generation adjust practice as capabilities evolve (Bengio et al., 2009; Huang et al., 2026a). Self-training therefore needs accessible practice, useful actions within that experience, and new tasks as the learner improves.

We propose VERA, which uses verified designs both to construct recovery practice and to seed the search for stronger specifications (Figure 2). Starting from a limited set of seeds, which may be externally supplied, **certified reduction** changes structure, connectivity, and parameters to expose construction, repair, and tuning decisions. It reduces the remaining decision burden through the starting state, normally retaining the seed's full specifications. Each task has a certified recovery path within the learner's edit interface and budget. The learner finds its own solution; verified progress, repair, and completion actions from successful rollouts supply supervised fine-tuning (SFT) targets.

After each weight update, **target expansion** uses the improved learner to search for circuits satisfying stronger specifications while preserving existing constraints. Fully verified designs become

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.

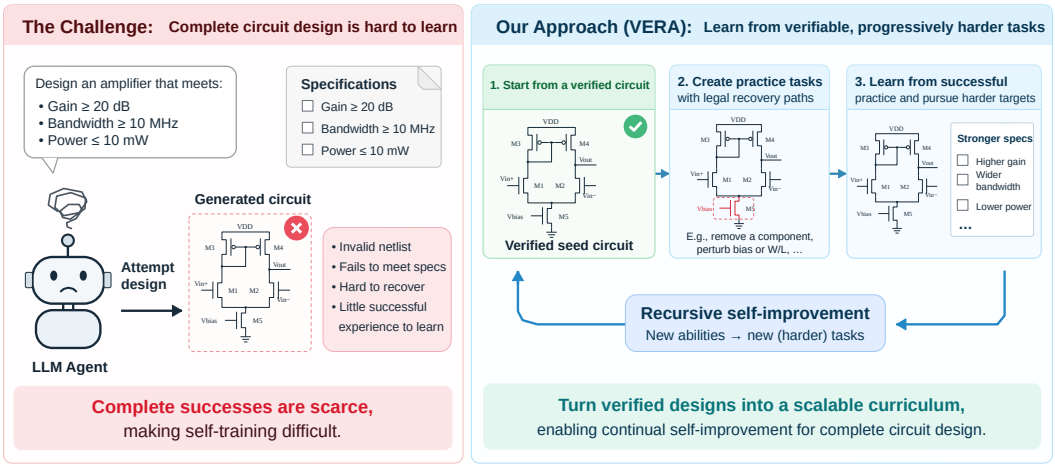


Figure 1: Motivation for VERA. Complete-design attempts can fail before reaching useful practice. Verified circuits provide accessible recovery tasks from which the learner can improve. Schematics and values are illustrative.

seeds for a fresh recovery-task bank, and learner outcomes guide subsequent perturbations and combinations. This *recursive self-improvement* (RSI) loop connects model improvement to the construction of subsequent training tasks. Construction and verification rules remain fixed, and the loop requires no further teacher solutions or manual task authoring after initialization.

We study VERA in the SPICE simulation environment on operational transconductance amplifiers (OTA), inverter chains (INV), and comparators (CMP). After one SFT update on programmatically generated tasks, the 9B learner's held-out complete-design success rises from 0/180 to 77/180 (42.8%), with gains on two external circuit benchmarks. We further examine how performance changes with training-data scale and whether renewed tasks support additional learning across updates.

Our contributions are:

- Certified task reduction that controls the starting state and remaining decision burden, converting verified seeds into focused and combined exercises with certified recovery paths within the learner's edit interface and budget.
- Action-level supervision that selects verified progress, repair, and completion steps from successful learner rollouts, reaching 42.8% complete-design success versus 26.1% for success-trajectory SFT on the same successful episodes.
- An RSI loop in which updated solvers discover stronger-specification seeds and construct a fresh task bank each round, reaching 68.3% success after three updates and exceeding fixed-bank and frozen-search controls.

2 Related Work

Circuit agents and simulation-grounded training. Circuit agents use simulation feedback (Lai et al., 2025; Kochar et al., 2025; Liu & Chitnis, 2025; Ahmadzadeh et al., 2025; Yu et al., 2026; Bujana & Karsilayan, 2026; Li & Kim, 2026; Hao et al., 2026), experience memory (Wang et al., 2025; Bao et al., 2026), multimodal revision with Bayesian sizing (Lai et al., 2026; Pan et al., 2026), and learned topology generation (Vijayaraghavan et al., 2025; Zhang et al., 2024; Islam et al., 2026; Kochar et al., 2026; Uhlich et al., 2026). Related systems address netlist repair (Feng et al., 2026), retrieval-assisted multi-agent design (Habib et al., 2026), cross-modal retrieval (Wang et al., 2026), and simulation-grounded training and evaluation (Govindugari, n.d.; Noori Zadeh & Elamien, 2026). With supplied topologies and search spaces, evolutionary methods (Liu et al., 2009), Bayesian optimization (Gu et al., 2024), and reinforcement learning (Settaluri et al., 2020; Choi et al., 2024; Brempong et al., 2026) support sizing; AnalogGym compares sizing performance and cost across

amplifier topologies (Li et al., 2025). VERA focuses on training-practice construction: verified circuits supply recovery tasks when complete-design successes are scarce, and selected learner actions supervise structural edits, repair, and tuning.

Self-training and recursive task generation. STaR and ReST$^{\text{EM}}$ alternate generation, correctness-based selection, and training (Zelikman et al., 2022; Singh et al., 2024). Absolute Zero couples task proposal and solving through executable feedback (Zhao et al., 2025a; Wilf et al., 2025; Liu et al., 2026a); R-Zero co-evolves a challenger and solver (Huang et al., 2026a; Liu et al., 2025; Lu et al., 2025; Li et al., 2026). Skill Self-Play evolves a skill library alongside task generation (Huang et al., 2026b). STOP improves scaffolding without weight updates (Zelikman et al., 2024; Agrawal et al., 2026; Yin et al., 2025; Zhang et al., 2025; 2026; Xia et al., 2025; Srikanth et al., 2026). In VERA, the updated learner discovers stronger-specification seeds for subsequent recovery tasks under fixed proposal and certification rules.

Curriculum learning and verifiable task construction. Curriculum learning organizes practice (Bengio et al., 2009; Chen et al., 2025; Park et al., 2026; Cowsik et al., 2026); reverse curricula and Backplay move starts from goals or along demonstrations (Florensa et al., 2017; Resnick et al., 2022). LADDER generates simpler variants (Simonds & Yoshiyama, 2025; Chen et al., 2026; Liu et al., 2026b; Sancaktar et al., 2026), while dataset decomposition uses teacher-validated subproblems (Zhao et al., 2025b). Asymmetric self-play asks agents to reverse or repeat actions (Sukhbaatar et al., 2018; Jana et al., 2026); Self-Play SWE-RL couples bug injection and repair with execution-based validation (Wei et al., 2026). BenchEvolver evolves reference solutions to construct harder coding tasks with executable validation (Wu et al., 2026). VERA constructs editable circuit starts with certified recovery paths, coupling selected-action training to the discovery of stronger-specification seeds for new practice.

3 VERA: Verifiable Escalation and Reduction

VERA controls both where a task starts and what its solution must satisfy. Reduction modifies the starting circuit while normally retaining the seed's specifications, creating certified recovery practice. Selected actions from successful recovery rollouts update the learner, which then searches for stronger-specification seeds to construct the next task bank (Figure 2).

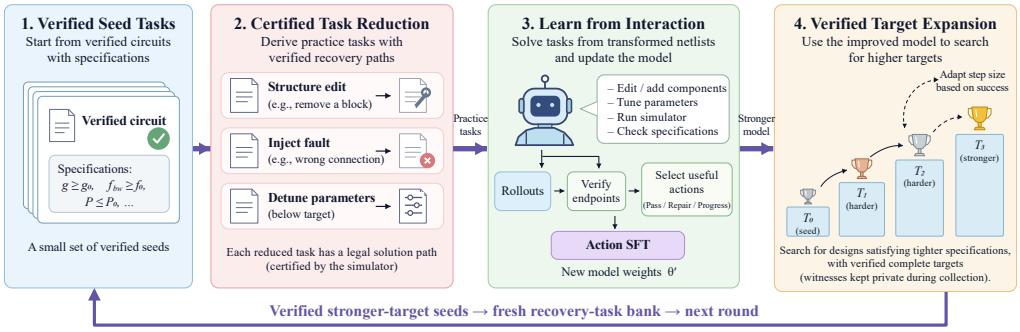


Figure 2: The VERA loop: verified seeds yield certified recovery tasks; selected actions from successful learner rollouts train the learner, which searches for stronger-specification seeds to build the next task bank. Bootstrap uses the initial bank before expansion.

3.1 Circuit Design and Seed Initialization

In the SPICE simulation environment, each task specifies a starting netlist, requirements, permitted edits, validity conditions (guards), and an interaction budget. The learner revises topology and sizing from SPICE feedback. One candidate must satisfy all assigned requirements under fixed testbenches, device models, and evaluators.

VERA constructs recovery tasks from verified seed circuits and uses learner rollouts for SFT. A *seed circuit* satisfies its associated specifications under the fixed verifier; a *starting circuit* is the state supplied for a particular task. Our initialization uses three human-constructed seed circuits, one each for OTA, INV, and CMP. Circuits supplied by another model or the learner can also enter the bank directly once verified under the same specifications and environment. Seed-derived starting circuits provide selected prerequisites for focused practice.

3.2 Certified Task Reduction

Reduction constructs task starting states through controlled edits to verified seeds. A fixed operator library defines supported edit sites, perturbation ranges, and permitted learner edits. A seeded pseudorandom sampler selects concrete sites and values for programmatic task construction. For an amplifier, deleting a bias branch creates a construction/modification task; changing a connection creates a diagnosis/repair task; detuning device dimensions creates a parameter-optimization task; combining these changes creates a multi-step task. Recovery normally retains the seed's full specifications. Explicitly local tasks may use narrower acceptance criteria while preserving mandatory guards.

Each candidate task is checked before collection. The starting circuit must fail its assigned goal, and a recorded recovery sequence must restore a passing circuit using the learner's edit interface within its budget, satisfying applicable path and endpoint guards. Certification establishes feasibility, not learner success. The seed and recovery record remain private during recovery-task rollouts: the learner receives the task and starting circuit, and must find its own solution through edits and SPICE feedback. Any permitted solution satisfying the task is accepted.

After bootstrap, learner outcomes guide subsequent task construction. Difficult combinations are separated into individually certified changes, or individual perturbations are weakened. Focused tasks practice component decisions, while combinations require their coordination across multiple edits and simulations.

3.3 Learning from Selected Actions

A *rollout* records the learner's edits and tool feedback. We retain only rollouts whose final candidate passes the assigned task. This endpoint check establishes eventual success; local action selection determines which intermediate decisions receive supervision. We compare each edit with the preceding evaluated circuit, using the supplied start for the first edit. *Pass* completes the task; *Repair* restores required function; *Progress* improves a violated target without worsening any other target violation. Selection requires verifier feedback establishing eligibility and satisfaction of all applicable guards; Progress additionally requires both circuits to function.

For a metric m_j with feasible interval $[\ell_j, u_j]$, define its violation as $v_j(c) = \max\{\ell_j - m_j(c),\, 0,\, m_j(c) - u_j\}$, omitting absent bounds for one-sided constraints. Let c^- and c^+ be the evaluated circuits before and after an edit. In addition to functionality and guard checks, Progress requires

$$v_j(c^+) \le v_j(c^-) \quad \forall j, \qquad v_j(c^+) < v_j(c^-) \quad \text{for at least one } j. \tag{1}$$

The task bank records assignments and provenance; the training set contains selected learner actions. The constructor's private recovery sequence certifies feasibility, while the learner's own decisions supply supervision. Each retained turn contributes (s_t, a_t), where s_t contains only the interaction history available before action a_t. Earlier actions, including unsuccessful attempts, and tool feedback remain context with zero loss; future feedback is excluded. We perform LoRA-based SFT (Hu et al., 2022), applying next-token cross-entropy loss only to the selected action's tokens. The selected-action pool $\mathcal{D}_k$ for round k is partitioned into $\mathcal{D}_k^{\text{train}}$ and $\mathcal{D}_k^{\text{val}}$ under Appendix A.3; only $\mathcal{D}_k^{\text{train}}$ contributes to weight updates.

Figure 3 shows a measured recovery trajectory for an inverter, which flips a binary input. Action a_1 reduces both switching delays and qualifies as Progress before the task is solved. Actions a_2–a_6 improve the already-feasible fall delay but worsen the violated rise delay; a_7 worsens both. Action a_8 passes both requirements. Only a_1 and a_8 receive supervision; the intervening edits remain context. A successful endpoint therefore makes the trajectory eligible without making every action a training target.

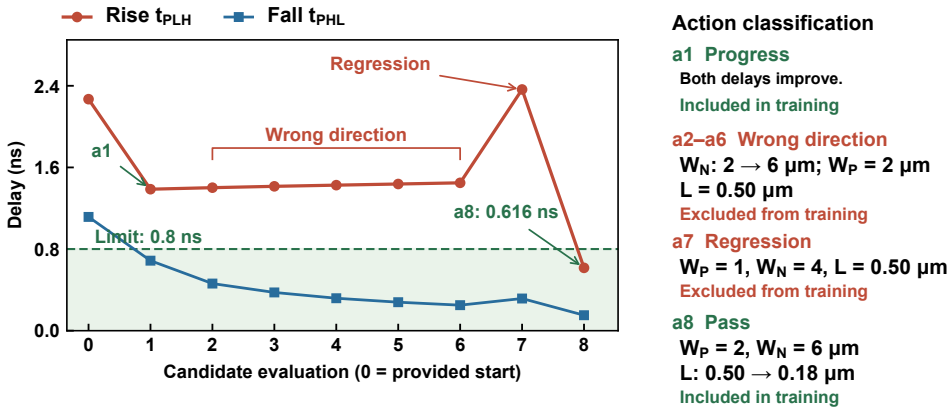


Figure 3: Inverter action selection. The learner tunes transistor widths W_P, W_N and channel length L (in μm). Both propagation delays, from an input change to the output rising (t_{PLH}) or falling (t_{PHL}), must fall below 0.8 ns. Left: simulated delays after each edit. Right: parameter changes and selection decisions.

3.4 Target Expansion and Recursive Improvement

Target expansion connects each weight update to the supply of subsequent training tasks. The updated learner searches from an existing verified seed for a circuit satisfying tighter requirements, for example, increasing amplifier gain while preserving bandwidth, stability, power, and operating conditions. Verified discoveries seed the next recovery-task bank.

The proposer polls requirements in a fixed order and adapts step sizes using direct-search principles (Kolda et al., 2003). It skips proposals already satisfied by the starting design, increases the step after verified success, and decreases it after unsuccessful polls, subject to configured limits. A failed minimum-step poll or exhausted budget ends the search; unresolved targets can be retried after learning. Only a circuit satisfying the stronger specification and all retained constraints becomes a new seed. The constructor applies the same certified transformations to this seed, producing recovery tasks under its new specifications. Appendix B describes the search and stopping rules.

Each round uses one fixed checkpoint θ_k for search and collection. During bootstrap, θ_0 collects from a fixed recovery-task bank derived from the initial seeds, without target expansion or adaptive task renewal. The selected actions train θ_1. Each subsequent round expands the targets with the updated learner and replaces the task bank with recovery tasks constructed from seeds verified under those targets. Training uses only the current round's collected actions, while prior learning carries forward through the model weights. Testbenches, device models, verifiers, edit interfaces, target ordering, and construction rules remain fixed. Failed searches retain seeds for later search; an empty selected-action set stops the update. Appendix B.3 and Algorithm 1 describe the update procedure.

4 Experiments

4.1 Setup and Evaluation Protocol

We evaluate VERA in the SPICE simulation environment, using ngspice (The ngspice Development Team, n.d.) and SKY130 (The SkyWater PDK Authors, n.d.) for construction, repair, parameter tuning, and multi-step tasks across OTA, INV, and CMP circuits. The learner is Qwen3.5-9B (Qwen Team, 2026). Internal success requires all assigned constraints in one candidate at the TT process corner.

The internal evaluation comprises 18 human-prepared held-out complete-design tasks: six per circuit family, evenly divided across easy, medium, and hard difficulty. Each task receives ten rollouts, each limited to 20 candidate evaluations and 60 model turns. Complete-design evaluation starts from a supplied weak netlist and requires a single candidate to satisfy all assigned specifications. All six tasks within each circuit family share the same D01 starter; Appendix A specifies the starting

Table 1: Main results on internal held-out tasks and external benchmarks. Entries report successful/total rollouts; internal overall SR is shown in parentheses (%).

	Held-out complete design (internal)				External transfer	
Model / training condition	Easy	Medium	Hard	All 18 (SR)	PhyChip v2	AnalogCoder Pro
Qwen3.5-9B base ($k = 0$)	0/60	0/60	0/60	0/180 (0.0)	8/150	30/60
+ Complete-task action SFT ($k = 1$)	0/60	1/60	0/60	1/180 (0.6)	35/150	37/60
+ Success-trajectory SFT ($k = 1$)	27/60	14/60	6/60	47/180 (26.1)	58/150	32/60
+ **VERA** ($k = 1$)	43/60	22/60	12/60	77/180 (42.8)	83/150	47/60
+ VERA ($k = 2$)	49/60	36/60	24/60	109/180 (60.6)	89/150	49/60
+ VERA ($k = 3$)	48/60	43/60	32/60	123/180 (68.3)	94/150	53/60

circuits, permitted edits, and test conditions. The 18 evaluation tasks are excluded from training, task construction, and checkpoint selection. Unless otherwise stated, model comparisons use identical task starts, testbenches, device models, verifiers, and interaction budgets.

External evaluation covers 50 PhyChip-v2 (Govindugari, n.d.) tasks and a 20-task AnalogCoder-Pro (Lai et al., 2026) subset, with three generations per task (Appendix D). Appendix A.3 gives the shared training and reporting protocol.

4.2 Main Results and Benchmark Transfer

Table 1 reports overall success rate (SR). After one update, VERA reaches 77/180 (42.8%) on held-out complete-design tasks, compared with 47/180 (26.1%) for Success-trajectory SFT. Both methods use the same successful recovery-task trajectories. VERA selects 1,200 action samples, while Success-trajectory SFT supervises all 3,850 assistant actions in these episodes (Table 12). These counts refer to the pools before the shared 90:10 split. Complete-task action SFT uses separate complete-design attempts and retains useful actions from failed episodes. Appendix E.1 reports equal-data and equal-collection-rollout comparisons. PhyChip-v2 results combine L1 SPICE execution checks with L2/L3 reference-based performance grading.

4.3 Complete-Design Performance by Difficulty

Figure 4 shows first-update gains across all circuit families and difficulty levels from a zero-success base learner. VERA reaches 65–80% on easy tasks and remains nonzero on hard tasks in every family. On the same tasks, GPT-5.6 Terra (OpenAI, 2026) and Grok-4.6 (SpaceXAI, 2026) achieve 100% on easy, 95–100% on medium, and 60–90% on hard specifications. Thus, recovery training supports complete design beyond easy tasks, while a gap to stronger models remains on solvable harder specifications.

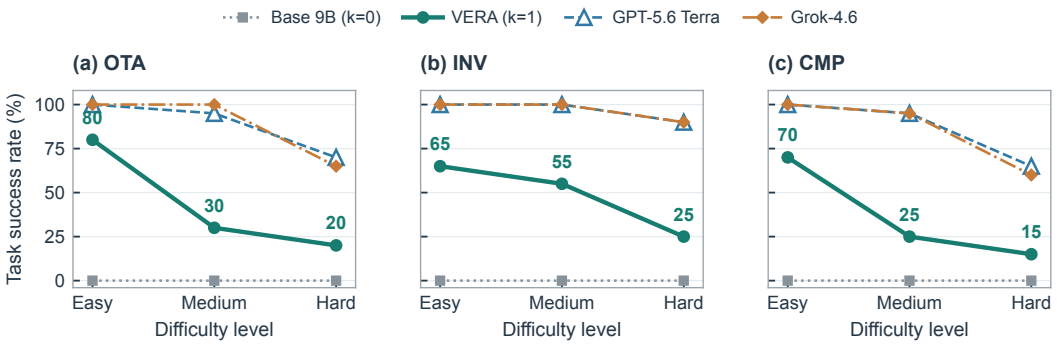


Figure 4: Complete-design success by circuit family and difficulty.

4.4 Recovery Experience: Collection and Learning Value

Starting from one verified seed circuit for each family, we construct recovery tasks by removing structure, changing connections, perturbing parameters, or combining these operations. The initial bank contains 26 subtasks per family: six construction/modification, eight diagnosis/repair, six parameter-optimization, and six multi-step tasks. Appendices E.2 and E.4 evaluate early target expansion and subsequent task renewal.

The base 9B model performs six rollouts per subtask. Across 78 subtasks and 468 attempts, it produces 213 successful trajectories (45.5%), with 77/156 successes on OTA, 105/156 on INV, and 31/156 on CMP (Table 2). These trajectories yield a pool of 1,200 selected action samples. A random sample-level 90:10 split assigns 1,080 actions to training and 120 to validation (Appendix A.3). This bank supplies successful practice for the same base model that fails the held-out complete-design tasks.

Table 2: Base 9B success on the 78 training subtasks. Entries report successful/attempted rollouts.

Task group	Tasks/family	OTA	INV	CMP	Total
Construction / modification	6	10/36	31/36	3/36	44/108
Diagnosis / repair	8	18/48	25/48	9/48	52/144
Parameter optimization	6	27/36	16/36	3/36	46/108
Multi-step design	6	22/36	33/36	16/36	71/108
Overall	**26**	**77/156**	**105/156**	**31/156**	**213/468**

Certification establishes a legal solution path; the observed 45.5% collection success shows that the current learner can also solve many of these tasks and supply training experience.

Figure 5 reports controls on the learning value of this experience. Recovery-task supervision raises easy complete-design SR from 0% to 15% with a 160-action pool per source, and from 0% to 20% at 90 learner-collection rollouts per source. Within the same recovery-trajectory pool, full selection reaches 71.7%, versus 45.0% for success-trajectory SFT and 51.7% without endpoint verification. Tables 11 and 12 give the corresponding results.

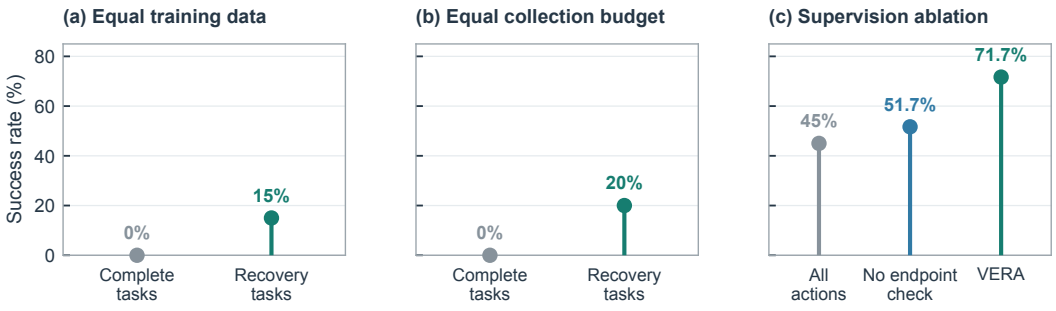


Figure 5: Controlled comparisons of recovery-task supervision: (a) equal action-pool size, (b) equal learner-collection budget, and (c) supervision ablation. All panels report SR on six easy complete-design tasks.

Recovery-task supervision retains an advantage when action-pool size or collection-rollout count is held fixed separately. Endpoint verification and action selection thus contribute separately (Figure 3).

4.5 Data Scaling and Recursive Improvement

Figure 6(a,b) tests one-update learning from nested subsets containing 12.5%, 25%, 50%, and 100% of the 1,080-sample training partition. All runs start from the same base checkpoint. Panel (a) evaluates the six easy complete-design tasks; panel (b) evaluates the external benchmarks. Increasing

the training set from 135 to 1,080 actions raises easy-task SR from 13.3% to 71.7%, with increasing scores on both external benchmarks. On AnalogCoder-Pro, the 135-action subset yields 26/60 successes, below the base model's 30/60; the 1,080-action subset reaches 47/60 (Tables 1 and 10).

Panel (c) compares full recursion, fixed-bank recollection, and a frozen search model. All branches continue from the same first-update checkpoint and use the shared learner architecture and LoRA configuration. At the third update, full recursion reaches 68.3% complete-design SR, compared with 53.9% for fixed-bank recollection and 57.2% for frozen search. The three branches share the learner, LoRA configuration, training procedure, and evaluation; they differ in task supply and, as a consequence, in the volume of collected experience. By the third update, cumulative learner collection comprises 1,566 rollouts for full recursion, 1,404 for fixed-bank recollection, and 1,446 for frozen search, excluding task construction and certification. Appendix E.4 reports the corresponding selected-action pool sizes. Tables 10 and 13 give the plotted results.

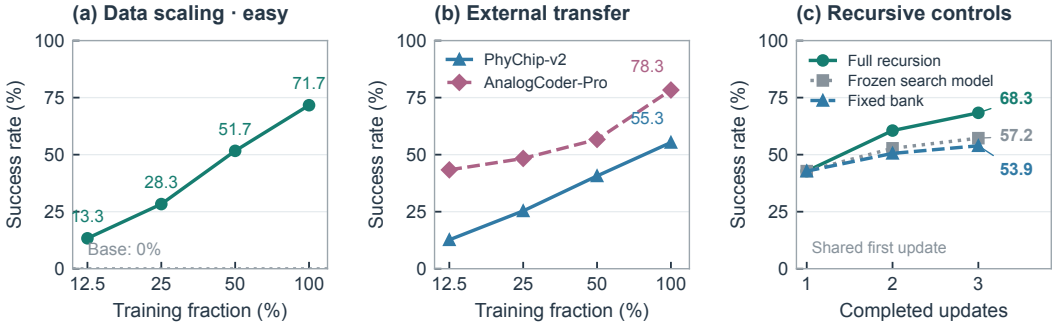


Figure 6: Data scaling and recursive controls: (a) easy-task scaling, (b) external transfer, and (c) task-renewal controls on all 18 complete-design tasks.

After bootstrap, full recursion and frozen search replace the recovery-task bank each round using seeds verified under that round's expanded targets. Earlier tasks and action pools are excluded from the new training set. In Algorithm 1, $\mathcal{Q}_1$ supplies the update to θ_2, and $\mathcal{Q}_2$ supplies the update to θ_3. Appendix E.4 reports each update's selected-action pool and cumulative collection cost.

All branches use the current learner for fresh collection and subsequent SFT. Table 3 defines the search model and task bank for each branch.

Table 3: Search and collection in the recursive comparison branches.

Branch	Target-search model	Recovery-task bank	Collection learner
Fixed bank	No target search	Fixed Q_0; fresh rollouts	Updated
Full recursion	Current checkpoint	Fresh bank from this round's stronger-specification seeds	Updated
Frozen search	Fixed θ_0	Fresh bank from this round's stronger-specification seeds	Updated

4.6 Task Construction for Bootstrapping

Table 4 compares target expansion before the first update (V) and recovery-task construction (H), reporting usable action yield and downstream success. Complete-task collection yields 160 usable actions from 90 learner rollouts, whereas recovery-task collection yields 1,200 from 468 rollouts. Figure 5(b) separately compares the sources at equal learner-collection budget; Appendix E.2 defines the settings.

Recovery and expansion address different learning needs. Recovery changes the starting state and remaining decision burden; expansion raises the final performance requirements. In the reported recovery-task pipelines, early expansion yields fewer usable actions (1,120 versus 1,200), reduces internal SR from 42.8% to 35.0%, and lowers both external scores. These observations suggest

that stronger requirements alone do not provide better bootstrap experience. They support the order used in VERA: first train on accessible recovery tasks, then use the improved learner to search for stronger targets and construct the next round's practice.

Table 4: First-update task-construction settings. V enables target expansion before training; H enables recovery-task construction.

V	H	First-update setting	Usable actions	Internal SR	PhyChip-v2 SR	AnalogCoder-Pro SR
Off	Off	Complete-task action SFT	160	0.6%	23.3%	61.7%
On	Off	Complete-task training + expansion	130	1.7%	20.0%	58.3%
Off	On	**VERA bootstrap (ours)**	**1,200**	**42.8%**	**55.3%**	**78.3%**
On	On	VERA bootstrap + early expansion	1,120	35.0%	48.7%	71.7%

Note: Usable actions are selected samples before splitting. Off/Off and Off/On reuse the corresponding experiments in Table 1. Internal SR covers all 18 tasks.

5 Discussion

From circuit practice to recursive learning. At $k = 3$, full recursion exceeds fixed-bank recollection by 14.4 percentage points and frozen search by 11.1 points. Together with the recovery-data controls, these pipeline comparisons support coupling learner updates to subsequent task construction under a fixed verifier.

Task coverage and conditions for transfer. Both external circuit benchmarks improve after bootstrap and from $k = 1$ to $k = 3$: PhyChip-v2 rises from 83/150 to 94/150 and AnalogCoder-Pro from 47/60 to 53/60. On PhyChip-v2's performance-graded L2/L3 tasks, first-update success rises from 0/66 to 11/66 (Appendix D), alongside improvements in L1 execution success. These results support transfer within circuit design. Appendix F describes the conditions for applying the construction to other domains.

Where does recursive improvement occur? Table 1 shows that most gains after bootstrap occur on harder tasks. From $k = 1$ to $k = 3$, medium-task successes rise from 22/60 to 43/60 and hard-task successes from 12/60 to 32/60, while easy-task successes rise from 43/60 to 48/60. Medium and hard tasks therefore account for 41 of the 46 additional successes (89.1%). This concentration is consistent with the intended role of target expansion: providing harder tasks as the learner's capabilities increase.

6 Conclusion

VERA connects accessible circuit practice to the renewal of training tasks. Verified seeds, which may be externally supplied, provide starting points for certified recovery tasks. Action selection extracts progress, repair, and completion steps from successful learner rollouts for supervised fine-tuning. The updated learner then searches for stronger-specification seeds, from which the same constructor generates a fresh task bank. This loop links what the learner can solve now to the experience available for its next update under a fixed verifier.

On held-out complete-design tasks with supplied weak starting circuits, success rises from 0/180 to 77/180 (42.8%) after bootstrap and 123/180 (68.3%) after three updates. Recovery-data and action-selection controls support the learning value of the collected experience, while full recursion exceeds fixed-bank recollection and frozen search. Gains on two external circuit benchmarks further support transfer within circuit design. Together, these results show how verified designs can sustain useful practice when complete-design successes are initially scarce.

Future work should examine longer training sequences, variation across training seeds, and the full cost of constructing and collecting experience. Extending the approach beyond circuits will require reliable executable verification and task transformations that preserve a certified recovery path.

AI USE STATEMENT

We used generative AI tools for the following purposes.

Writing assistance and polishing. AI tools assisted with language editing, improving readability, and organizing the presentation. The authors reviewed the suggested revisions to ensure that they preserved the intended meaning.

Literature retrieval and discovery. AI tools assisted with finding relevant literature and organizing candidate references for the related-work discussion. The authors selected the cited works and reviewed their use in the manuscript.

Research ideation and execution. AI tools served as aids for discussing research ideas and supporting implementation. The authors led the research and made the final methodological decisions.

Drafting manuscript sections. AI tools assisted with drafting manuscript passages based on the study materials and author guidance. The authors reviewed and revised these drafts for consistency with the methods and results reported in the paper.

Synthetic training data generation. AI models were used to generate synthetic training data for circuit-design tasks. The generated data were evaluated and filtered using simulation feedback before training.

The authors reviewed the AI-assisted work and take responsibility for the final text, references, implementation, and scientific claims.

REPRODUCIBILITY STATEMENT

Our anonymous repository provides the implementation used in this work. The main text and appendices specify the task construction, verification, training, search, and evaluation procedures, including key budgets, data splits, and experimental settings. All reported circuit results are evaluated with fixed SPICE-based verifiers under the stated testbenches and device models.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alex Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. Gepa: Reflective prompt evolution can outperform reinforcement learning. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/0e9e708b6f48e14fd0ac29e167413f76-Paper-Conference.pdf.

Mohsen Ahmadzadeh, Kaichang Chen, and Georges G. E. Gielen. AnaFlow: Agentic LLM-based workflow for reasoning-driven explainable and sample-efficient analog circuit sizing. In *2025 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–7, 2025. doi: 10.1109/ICCAD66269.2025.11240818. URL https://doi.org/10.1109/ICCAD66269.2025.11240818.

Zhixuan Bao, Zhuoyi Lin, Jiageng Wang, Jinhai Hu, Yuan Gao, Yaoxin Wu, Xiaoli Li, and Xun Xu. Analogagent: Self-improving analog circuit design automation with llm agents. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, KDD '26, pp. 10419–10429, New York, NY, USA, 2026. Association for Computing Machinery. ISBN 9798400722592. doi: 10.1145/3770855.3818993. URL https://doi.org/10.1145/3770855.3818993.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum Learning. In *International Conference on Machine Learning*, pp. 41–48, 2009. doi: 10.1145/1553374.1553380. URL https://mlanthology.org/icml/2009/bengio2009icml-curriculum/.

Osei Brempong, Mohammed Ayman Habib, Vivan Poddar, and Morteza Fayazi. ORACLE: A multi-objective reinforcement learning-based analog circuit design optimizer with large language models-guided exploration, 2026. URL https://arxiv.org/abs/2608.04999.

Antonio J. Bujana and Aydin I. Karsilayan. A self-calibrating framework for analog circuit sizing using LLM-derived analytical equations, 2026. URL https://arxiv.org/abs/2604.07387.

Justin Chih-Yao Chen, Archiki Prasad, Zaid Khan, Joykirat Singh, Runchu Tian, Elias Stengel-Eskin, and Mohit Bansal. Cog-drift: Exploration on adaptively reformulated instances enables learning from hard reasoning problems, 2026. URL https://arxiv.org/abs/2604.04767.

Xiaoyin Chen, Jiarui Lu, Minsu Kim, Dinghuai Zhang, Jian Tang, Alexandre Piché, Nicolas Gontier, Yoshua Bengio, and Ehsan Kamalloo. Self-evolving curriculum for llm reasoning, 2025. URL https://arxiv.org/abs/2505.14970.

Youngchang Choi, Sejin Park, Minjeong Choi, Kyongsu Lee, and Seokhyeong Kang. Ma-opt: Reinforcement learning-based analog circuit optimization using multi-actors. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 71(5):2045–2056, 2024. doi: 10.1109/TCSI.2024.3356582.

Aditya Cowsik, Kfir Dolev, Michael Y. Li, G. Bruno De Luca, Nourya Cohen, Noah D. Goodman, and Yoav Levine. Self-play pretraining with zero data, 2026. URL https://arxiv.org/abs/2609.30063.

Zhe Feng, Yutong Ye, Jianxun Zhou, Wenbin Guo, Lihua Xu, Wendong Lu, Yunlai Zhu, Zuyu Xu, Zuheng Wu, and Yuehua Dai. Spicediff agent simulation efficient llm guided planning for specification driven analog circuit netlist repair. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2026. doi: 10.1109/TCAD.2026.3729375.

Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In Sergey Levine, Vincent Vanhoucke, and

Ken Goldberg (eds.), *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pp. 482–495. PMLR, 13–15 Nov 2017. URL https://proceedings.mlr.press/v78/florensa17a.html.

Nithin Reddy Govindugari. PhyChip: A verifiable-reward environment for analog circuit design. Public technical report and software repository, n.d. URL https://github.com/nithingovindugari/phychip/blob/3cb5e7400be4d2df1db0959a2bb8d07e3acca019/final_report/PhyChip_Technical_Report.md. Undated public technical report; pinned to commit 3cb5e7400be4d2df1db0959a2bb8d07e3acca019.

Tianchen Gu, Jiaqi Wang, Zhaori Bi, Changhao Yan, Fan Yang, Yajie Qin, Tao Cui, and Xuan Zeng. tss-bo: Scalable bayesian optimization for analog circuit sizing via truncated subspace sampling. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, 2024. doi: 10.23919/DATE58400.2024.10546770.

Mohammed Ayman Habib, Rylan Hart, and Morteza Fayazi. Aallm: An end-to-end analog circuit design framework from topology generation to sizing using large language models, 2026. URL https://arxiv.org/abs/2608.13472.

Yijia Hao, Pratibha Verma, Dongxu Guo, Cristian Sestito, Michael O'Boyle, Christos-Savvas Bouganis, and Themis Prodromakis. AgenticSizing: A large language model-based multi-agent framework for analog circuit sizing, 2026. URL https://arxiv.org/abs/2609.25873.

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 130770–130790, 2026a. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/d49b9aacebda61051166335af6fd3061-Paper-Conference.pdf.

Siyuan Huang, Pengyu Cheng, Haotian Liu, Tao Chen, Yihao Liu, Jingwei Ni, Shijie Zhou, Ziyi Yang, Gangwei Jiang, Mengyu Zhou, Yu Cheng, Xiaoxi Jiang, and Guanjun Jiang. Skill self-play: Pushing the frontier of llm capability with co-evolving skills, 2026b. URL https://arxiv.org/abs/2607.22529.

Md Touhidul Islam, Sujan Kumar Saha, Farimah Farahmandi, and Mark Tehranipoor. CircuitFormer: A circuit language model for analog topology design from natural language prompt, 2026. URL https://arxiv.org/abs/2605.05773.

Swadesh Jana, Cansu Sancaktar, Tomáš Daniš, Georg Martius, Antonio Orvieto, and Pavel Kolev. Gasp: Guided asymmetric self-play for coding llms, 2026. URL https://arxiv.org/abs/2603.15957.

Dimple Vijay Kochar, Hanrui Wang, Anantha P. Chandrakasan, and Xin Zhang. LEDRO: LLM-enhanced design space reduction and optimization for analog circuits. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, 2025. doi: 10.1109/ICLAD65226.2025.00011. URL https://doi.org/10.1109/ICLAD65226.2025.00011.

Dimple Vijay Kochar, Hae-Seung Lee, and Anantha P. Chandrakasan. Towards reliable AI-assisted analog design: Template-constrained LLM agents for SAR ADC generation, 2026. URL https://arxiv.org/abs/2607.14165.

Tamara G. Kolda, Robert Michael Lewis, and Virginia Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM Review*, 45(3):385–482, 2003. doi: 10.1137/S003614450242889. URL https://doi.org/10.1137/S003614450242889.

Yao Lai, Sungyoung Lee, Guojin Chen, Souradip Poddar, Mengkang Hu, David Z. Pan, and Ping Luo. AnalogCoder: Analog circuit design via training-free code generation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(1):379–387, 2025. doi: 10.1609/aaai.v39i1.32016. URL https://ojs.aaai.org/index.php/AAAI/article/view/32016.

Yao Lai, Souradip Poddar, Sungyoung Lee, Guojin Chen, Mengkang Hu, Bei Yu, Ping Luo, and David Z. Pan. Analogcoder-pro: Unifying analog circuit generation and optimization via multi-modal llms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2026. doi: 10.1109/TCAD.2026.3673493.

Fangyuan Li, Pengfei Li, Shijie Wang, Junqi Gao, Jianxing Liu, Biqing Qi, and Yuqiang Li. WIST: Web-grounded iterative self-play tree for domain-targeted reasoning improvement. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 31563–31585, July 2026. doi: 10.18653/v1/2026.acl-long.1456. URL https://aclanthology.org/2026.acl-long.1456/.

Jintao Li, Haochang Zhi, Ruiyu Lyu, Wangzhen Li, Zhaori Bi, Keren Zhu, Yanhan Zeng, Weiwei Shan, Changhao Yan, Fan Yang, Yun Li, and Xuan Zeng. Analoggym: An open and practical testing suite for analog circuit synthesis. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '24, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400710773. doi: 10.1145/3676536.3697117. URL https://doi.org/10.1145/3676536.3697117.

Xunqi Li and Chris H. Kim. SABLE: An NDA-safe closed-loop LLM framework for analog circuit optimization in industrial EDA flows, 2026. URL https://arxiv.org/abs/2607.03701.

Bo Liu, Yan Wang, Zhiping Yu, Leibo Liu, Miao Li, Zheng Wang, Jing Lu, and Francisco V. Fernández. Analog circuit optimization system based on hybrid evolutionary algorithms. *Integration*, 42(2):137–148, 2009. ISSN 0167-9260. doi: https://doi.org/10.1016/j.vlsi.2008.04.003. URL https://www.sciencedirect.com/science/article/pii/S0167926008000126.

Bo Liu, Chuanyang Jin, Seungone Kim, Weizhe Yuan, Wenting Zhao, Ilia Kulikov, Xian Li, Sainbayar Sukhbaatar, Jack Lanchantin, and Jason Weston. SPICE: Self-play in corpus environments improves reasoning, 2025. URL https://arxiv.org/abs/2510.24684.

Bo Liu, Simon Yu, Yiding Jiang, Ao Qu, Andrew Zhao, Zichen Liu, Junsu Kim, Zijian Zhou, Seungone Kim, Tongzheng Ren, Mickel Liu, Hanfei Yu, Zhaorun Chen, Weiyan Shi, Paul Pu Liang, Luke Zettlemoyer, Yejin Choi, and Natasha Jaques. SPADE: Self-play in adaptive synthetic executable environments, 2026a. URL https://arxiv.org/abs/2608.19197.

Chang Liu and Danial Chitnis. EEsizer: LLM-based ai agent for sizing of analog and mixed signal circuit, 2025. URL https://arxiv.org/abs/2509.25510.

Minghui Liu, Thomas Magelinski, Dehao Yuan, Qi Yu, and Furong Huang. Ladders of thought: A self-evolving curriculum of progressively simplified reasoning traces, 2026b. URL https://arxiv.org/abs/2609.25643.

Hongliang Lu, Yuhang Wen, Pengyu Cheng, Ruijin Ding, Haotian Xu, Jiaqi Guo, Chutian Wang, Haonan Chen, Xiaoxi Jiang, and Guanjun Jiang. Search self-play: Pushing the frontier of agent capability without supervision, 2025. URL https://arxiv.org/abs/2510.18821.

Danial Noori Zadeh and Mohamed B. Elamien. Analog-DB: An agent-first analog integrated circuit database, from blocks to systems, 2026. URL https://arxiv.org/abs/2609.01286.

OpenAI. GPT-5.6 Terra model. OpenAI API documentation, 2026. URL https://developers.openai.com/api/docs/models/gpt-5.6-terra. Accessed 2026-09-26.

Guanyuan Pan, Shuai Wang, Yugui Lin, Tiansheng Zhou, Pietro Liò, Zhenxin Zhao, and Yaqi Wang. VLM-CAD: Vision-language models for analog circuit design, 2026. URL https://arxiv.org/abs/2601.07315.

Chansung Park, Juyong Jiang, Fan Wang, Sayak Paul, Jiasi Shen, Jing Tang, and Jianguo Li. Tarot: Test-driven and capability-adaptive curriculum reinforcement fine-tuning for code generation with large language models, 2026. URL https://arxiv.org/abs/2602.15449.

Qwen Team. Qwen3.5-9B model card. Hugging Face, 2026. URL https://huggingface.co/Qwen/Qwen3.5-9B. Accessed 2026-09-26.

Cinjon Resnick, Roberta Raileanu, Sanyam Kapoor, Alexander Peysakhovich, Kyunghyun Cho, and Joan Bruna. Backplay: "man muss immer umkehren", 2022. URL https://arxiv.org/abs/1807.06919.

Cansu Sancaktar, David Zhang, Gabriel Synnaeve, and Taco Cohen. A deep dive into scaling rl for code generation with synthetic data and curricula, 2026. URL https://arxiv.org/abs/2603.24202.

Keertana Settaluri, Ameer Haj-Ali, Qijing Huang, Kourosh Hakhamaneshi, and Borivoje Nikolic. Autockt: Deep reinforcement learning of analog circuit designs. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 490–495, 2020. doi: 10.23919/DATE48585.2020.9116200.

Toby Simonds and Akira Yoshiyama. Ladder: Self-improving llms through recursive problem decomposition, 2025. URL https://arxiv.org/abs/2503.00735.

Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron T Parisi, Abhishek Kumar, Alexander A Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Fathy Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura A Culp, Lechao Xiao, Maxwell Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL https://openreview.net/forum?id=lNAyUngGFK. Expert Certification.

SpaceXAI. Introducing Grok 4.6. Official model announcement, August 2026. URL https://x.ai/news/grok-4-6. Published August 12, 2026; accessed 2026-09-26.

Dhruv Srikanth, Bingchen Zhao, Dixing Xu, Yuxiang Wu, and Zhengyao Jiang. Recursive self-improvement of AI research agents, 2026. URL https://arxiv.org/abs/2609.26457.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=SkT5Yg-RZ.

The ngspice Development Team. ngspice documentation. Official software and documentation, n.d. URL https://ngspice.sourceforge.io/docs.html. Accessed 2026-09-26.

The SkyWater PDK Authors. SkyWater SKY130 PDK documentation. Official process design kit documentation, n.d. URL https://skywater-pdk.readthedocs.io/en/main/. Accessed 2026-09-26.

Stefan Uhlich, Yağız Gençer, Andrea Bonetti, Arun Venkitaraman, Chia-Yu Hsieh, Eisaku Ohbuchi, and Lorenzo Servadei. Spicing up genetic netlist generation with LLMs. In *2026 ACM/IEEE International Symposium on Machine Learning for CAD (MLCAD)*, 2026. doi: 10.1145/3831599.3840324. URL https://doi.org/10.1145/3831599.3840324.

Prashanth Vijayaraghavan, Luyao Shi, Ehsan Degan, Vandana Mukherjee, and Xin Zhang. AUTOCIRCUIT-RL: Reinforcement learning-driven LLM for automated circuit topology generation. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 61498–61512. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/vijayaraghavan25a.html.

Yihan Wang, Lei Li, Yao Lai, Jing Wang, and Yan Lu. Analogretriever: Learning cross-modal representations for analog circuit retrieval, 2026. URL https://arxiv.org/abs/2604.23195.

Zining Wang, Jian Gao, Weimin Fu, Xiaolong Guo, and Xuan Zhang. Analogsage: Self-evolving analog design multi-agents with stratified memory and grounded experience, 2025. URL https://arxiv.org/abs/2512.22435.

Yuxiang Wei, Zhiqing Sun, Emily McMilin, Jonas Gehring, David W. Zhang, Gabriel Synnaeve, Daniel Fried, LINGMING ZHANG, and Sida Wang. Toward training superintelligent software agents through self-play SWE-RL. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=0ophJB76sC.

Alex Wilf, Pranjal Aggarwal, Bryan Parno, Daniel Fried, Louis-Philippe Morency, Paul Pu Liang, and Sean Welleck. Propose, solve, verify: Self-improving language models with executable feedback, 2025. URL https://arxiv.org/abs/2512.18160.

Yangzhen Wu, Aaron J. Li, Wenjie Ma, Li Cao, Ziheng Zhou, Mert Cemri, Shu Liu, Yuran Xiu, Chenxiao Yan, Haikun Zhao, Bin Yu, Ion Stoica, and Dawn Song. Benchevolver: Frontier task synthesis via solution-centric evolution, 2026. URL https://arxiv.org/abs/2606.01286.

Chunqiu Steven Xia, Zhe Wang, Yan Yang, Yuxiang Wei, and Lingming Zhang. Live-SWE-agent: Can software engineering agents self-evolve on the fly?, 2025. URL https://arxiv.org/abs/2511.13646.

Xunjian Yin, Xinyi Wang, Liangming Pan, Li Lin, Xiaojun Wan, and William Yang Wang. Gödel Agent: A self-referential agent framework for recursively self-improvement. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 27890–27913, July 2025. doi: 10.18653/v1/2025.acl-long.1354. URL https://aclanthology.org/2025.acl-long.1354/.

Xi Yu, Dmitrii Torbunov, Soumyajit Mandal, and Yihui Ren. AutoSizer: Automatic sizing of analog and mixed-signal circuits via large language model (LLM) agents, 2026. URL https://arxiv.org/abs/2602.02849.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances in Neural Information Processing Systems*, volume 35, pp. 15476–15488. Curran Associates, Inc., 2022. doi: 10.52202/068431-1126. URL https://proceedings.neurips.cc/paper_files/paper/2022/file/639a9a172c044fbb64175b5fad42e9a5-Paper-Conference.pdf.

Eric Zelikman, Eliana Lorch, Lester Mackey, and Adam Tauman Kalai. Self-taught optimizer (STOP): Recursively self-improving code generation. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=46Zgqo4QIU.

Haoyi Zhang, Shizhao Sun, Yibo Lin, Runsheng Wang, and Jiang Bian. AnalogXpert: Automating analog topology synthesis by incorporating circuit design expertise into large language models, 2024. URL https://arxiv.org/abs/2412.19824.

Jenny Zhang, Shengran Hu, Cong Lu, Robert Lange, and Jeff Clune. Darwin Gödel machine: Open-ended evolution of self-improving agents, 2025. URL https://arxiv.org/abs/2505.22954.

Jenny Zhang, Bingchen Zhao, Wannan Yang, Jakob Foerster, Jeff Clune, Minqi Jiang, Sam Devlin, and Tatiana Shavrina. Hyperagents, 2026. URL https://arxiv.org/abs/2603.19461.

Andrew Zhao, Yiran Wu, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 105816–105879. Curran Associates, Inc., 2025a. doi: 10.52202/

085713-3534. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/9837dc00ff67d176373268ed48042d49-Paper-Conference.pdf.

Wanru Zhao, Lucas Page-Caccia, Zhengyan Shi, Minseon Kim, Weijia Xu, and Alessandro Sordoni. Learning to solve complex problems via dataset decomposition. In D. Belgrave, C. Zhang, H. Lin, R. Pascanu, P. Koniusz, M. Ghassemi, and N. Chen (eds.), *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pp. 151390–151422. Curran Associates, Inc., 2025b. doi: 10.52202/085713-5064. URL https://proceedings.neurips.cc/paper_files/paper/2025/file/de3cb41a04a51723e92ef1b9c2e9dfc9-Paper-Conference.pdf.

A CIRCUIT ENVIRONMENT AND EVALUATION PROTOCOL

The SPICE simulation environment provides editable netlists, ngspice feedback, and a restricted circuit-edit interface for OTA, INV, and CMP tasks.

A.1 SIMULATION ENVIRONMENT

We use ngspice (The ngspice Development Team, n.d.) and the open-source SKY130 process design kit (The SkyWater PDK Authors, n.d.) for transistor-level OTA, INV, and CMP tasks. Agents edit netlist devices, dimensions, and connections, while testbenches, models, and evaluators remain fixed. SPICE supplies feedback and acceptance measurements. Internal complete-design evaluation uses the typical–typical (TT) corner.

A.2 TRAINING ENVIRONMENT

Because computational resources are limited, we use Qwen3.5-9B (Qwen Team, 2026) as the backbone. It is trained with action-level LoRA SFT (Hu et al., 2022) on one NVIDIA A100 80 GB GPU in BF16. Training and evaluation use distinct task instances within the same circuit families, with different device-parameter configurations and target specifications; topological similarity may remain.

The bootstrap training configuration uses two epochs over the training partition, LoRA rank $r = 64$, dropout 0.05, learning rate 1×10^{-5}, per-device batch size 2, and 8 gradient-accumulation steps, giving an effective batch size of 16. "Base 9B" denotes the released checkpoint, including its general post-training, before circuit-specific SFT.

A.3 SHARED TRAINING AND REPORTING PROTOCOL

Seed initialization and evaluation separation. The bootstrap uses the seed initialization and recovery-task bank described in Section 4.4. The 18 held-out complete-design tasks are excluded from training, task construction, and checkpoint selection.

Frozen conditions. The controlled studies use Qwen3.5-9B, the LoRA settings in Appendix A, and the same prompts, decoding, verifier, and single-episode interface except for the assigned variable. Internal evaluation follows Section 4.

Each condition uses one training run with random seed 42. Evaluation uses the checkpoint saved at the end of training; validation loss does not select the checkpoint.

Data and budget definitions. Action-pool size counts the action samples retained by each condition's supervision rule before splitting. Training amount counts unique samples supplied to SFT after splitting and subset selection. Each action pool is randomly split 90:10 at the sample level, without grouping by trajectory. Only the training partition contributes to parameter updates; the validation partition monitors loss. An epoch traverses the training set once. In the equal-amount control, M denotes the action-pool size before splitting. The collection-budget control fixes the number of learner rollouts. Appendix E.2 reports each pipeline's usable action yield. Experience-production cost includes seed creation, target search, certification, calibration, and collection attempts. Table 14 reports cumulative learner collection, excluding task construction and certification.

Result reporting. Rollout success is first averaged within each test task, and tasks are then averaged equally. Tables report point estimates on the stated fixed-denominator scale.

A.4 RELEASED INFERENCE INTERFACE

Dynamic tasks use the `TaskSpec` branch of `agent/src/analog_trace/runner.py`. The system prompt requests one tool call per response, edits within the declared scope, preservation of guards, and complete acceptance in one candidate. The user message contains the public task specification, initial circuit, and feedback, as constructed by `agent/src/analog_trace/engine.py`. Constructor recovery records remain private.

Table 5: Reference inference settings in the released code.

Setting	Value
Sampling	Temperature 1.0; maximum 4,096 output tokens per response
Top-p	Not explicitly set; service default
Tool use	Automatic tool choice; no provider retries for dynamic tasks

Table 5 follows `configs/provider.example.json`. The released runner connects to an externally served model and records its supplied `policy_version` with each round and trajectory.

A.5 Evaluation Methodology

Complete-design tasks. The internal held-out evaluation contains 18 complete-design tasks prepared in advance by humans: six each for OTA, INV, and CMP. Within each family, two tasks are designated easy, two medium, and two hard. Each model–task pair receives 10 rollouts. Each family–difficulty combination contains two tasks and contributes 20 rollouts. Each family and each difficulty level therefore contributes 60 rollouts, giving 180 rollouts overall. A rollout is limited to 20 circuit evaluations and 60 agent turns. Success requires all performance and resource constraints to hold in one valid candidate.

Initial circuits and fixed test conditions. All six complete-design tasks within each family use the same weak D01 starter, with the family-specific target specifications in Table 7. These are supplied circuit netlists requiring structural or performance improvement; complete-design success means satisfying the full specification from that starting circuit. The three starter netlists are supplied in `task_starts/`. Table 6 summarizes the circuits and fixed testbenches; all use a 1.8 V supply and the TT process corner.

Table 6: D01 starting circuits and fixed test conditions by family.

Tasks	Supplied starting circuit	Fixed load and input
OTA-D01–D06	PMOS differential pair, NMOS current-mirror load, and PMOS bias branch.	100 pF load; AC common mode 0.45 V; unity-gain transient input steps between 0.3 and 0.5 V.
INV-D01–D06	One complementary CMOS stage; PMOS/NMOS widths 0.84/0.42 μm and lengths 0.50 μm.	100 fF load; 0–1.8 V input pulses, 1 ns rise/fall times and 200 ns period; DC sweep 0–1.8 V.
CMP-D01–D06	NMOS differential pair, PMOS current-mirror load, and a 20 kΩ tail resistor.	50 fF load; $V_{\mathrm{INN}} = 0.9$ V, V_{INP} steps 0.88→0.92→0.88 V; delay uses the output falling crossing at 0.9 V.

Agents may resize devices, add or remove devices, and change connections or topology within each family's device rules; the testbench and PDK remain fixed. INV must end as exactly three complementary CMOS stages, with no independent R/C devices and output confined to $[-0.01, 1.81]$ V. Its static power, dynamic power, and MOS area limits are 50 μW, 80 μW, and 20 μm^2, respectively. OTA uses an agent-selected external bias of 1–40 μA on a 0.1 μA grid, power at most 3 mW, an area proxy at most 400, and CMRR and both PSRR values at least 10 dB; settling and overshoot are recorded but do not gate acceptance. CMP permits MOS devices, resistors, capacitors, and helper subcircuits, but no added sources or model/analysis directives; its task-specific power limits are listed in Table 7.

Synthesized subtasks. The recovery-task bank described in Section 4.4 spans construction/modification, diagnosis/repair, parameter optimization, and multi-step design. Initial SPICE feedback exposes symptoms; local or complete-specification predicates define acceptance through `stage_pass` or `full_pass`.

Complete-task action SFT control. The base 9B model performs 90 rollouts on nine additional complete-design tasks, three per circuit family, with ten rollouts per task. These tasks are distinct

from the evaluation instances and have comparable difficulty but different specification requirements. Actions that correct simulator-reported netlist errors or satisfy the local repair/progress criteria in Section 3 are retained, including actions from trajectories that do not reach a complete-design endpoint. The resulting adapter is trained with the configuration above and is denoted *Complete-task action SFT*.

The complete-task control obtains no successful episodes from its 90 rollouts but supplies 160 selected local actions. Appendix E.1 compares training sources at equal action amounts and learner-collection budgets; Appendix E.3 compares supervision choices.

The anonymous code repository provides the circuit environment, task construction and certification, target search, collection, and evaluation, with an interface for externally updated models.

A.6 Complete-Design Specifications and Per-Task Results

Table 7 lists the held-out task specifications and per-task results. After the first update, VERA achieves 26/60 successes on OTA, 29/60 on INV, and 22/60 on CMP; the base model achieves 0/60 in each family.

Table 7: Complete-design success on the 18 held-out tasks (successful rollouts out of 10).

		Successful rollouts / 10		
Task	Core performance	Qwen3.5 9B base	9B + Complete-task action SFT	**9B + VERA** (k=1)
Easy				
OTA-D01	Gain $\geq$ 40 dB; GBW $\geq$ 0.30 MHz; SR $\geq$ 0.15 V/μs; PM $\geq 40^\circ$	0/10	0/10	**8/10**
OTA-D02	Gain $\geq$ 35 dB; GBW $\geq$ 0.60 MHz; SR $\geq$ 0.30 V/μs; PM $\geq 40^\circ$	0/10	0/10	**8/10**
INV-D01	$V_{\mathrm{OH}} \geq 1.5$ V; $V_{\mathrm{OL}} \leq 0.3$ V; $\mathrm{NM_L}, \mathrm{NM_H} \geq 0.55$ V; $t_{\mathrm{pd,max}} \leq 0.45$ ns	0/10	0/10	**7/10**
INV-D02	$V_{\mathrm{OH}} \geq 1.5$ V; $V_{\mathrm{OL}} \leq 0.3$ V; $\mathrm{NM_L}, \mathrm{NM_H} \geq 0.88$ V; $t_{\mathrm{pd,max}} \leq 0.70$ ns	0/10	0/10	**6/10**
CMP-D01	$V_{\mathrm{OH}} \geq 1.20$ V; $V_{\mathrm{OL}} \leq 0.60$ V; $t_{\mathrm{decision}} \leq 20$ ns; $P \leq 1.0$ mW	0/10	0/10	**7/10**
CMP-D02	$V_{\mathrm{OH}} \geq 1.65$ V; $V_{\mathrm{OL}} \leq 0.15$ V; $t_{\mathrm{decision}} \leq 60$ ns; $P \leq 0.20$ mW	0/10	0/10	**7/10**
Medium				
OTA-D03	Gain $\geq$ 60 dB; GBW $\geq$ 0.70 MHz; SR $\geq$ 0.25 V/μs; PM $\geq 45^\circ$	0/10	0/10	**3/10**
OTA-D04	Gain $\geq$ 55 dB; GBW $\geq$ 1.00 MHz; SR $\geq$ 0.40 V/μs; PM $\geq 45^\circ$	0/10	0/10	**3/10**
INV-D03	$V_{\mathrm{OH}} \geq 1.5$ V; $V_{\mathrm{OL}} \leq 0.3$ V; $\mathrm{NM_L}, \mathrm{NM_H} \geq 0.70$ V; $t_{\mathrm{pd,max}} \leq 0.35$ ns	0/10	1/10	**5/10**
INV-D04	$V_{\mathrm{OH}} \geq 1.5$ V; $V_{\mathrm{OL}} \leq 0.3$ V; $\mathrm{NM_L}, \mathrm{NM_H} \geq 0.84$ V; $t_{\mathrm{pd,max}} \leq 0.50$ ns	0/10	0/10	**6/10**
CMP-D03	$V_{\mathrm{OH}} \geq 1.20$ V; $V_{\mathrm{OL}} \leq 0.60$ V; $t_{\mathrm{decision}} \leq 12$ ns; $P \leq 0.7$ mW	0/10	0/10	**3/10**
CMP-D04	$V_{\mathrm{OH}} \geq 1.65$ V; $V_{\mathrm{OL}} \leq 0.15$ V; $t_{\mathrm{decision}} \leq 40$ ns; $P \leq 0.15$ mW	0/10	0/10	**2/10**
Hard				
OTA-D05	Gain $\geq$ 80 dB; GBW $\geq$ 1.10 MHz; SR $\geq$ 0.35 V/μs; PM $\geq 50^\circ$	0/10	0/10	**2/10**
OTA-D06	Gain $\geq$ 75 dB; GBW $\geq$ 1.40 MHz; SR $\geq$ 0.50 V/μs; PM $\geq 50^\circ$	0/10	0/10	**2/10**

Continued on next page

Table 7: Complete-design success on the 18 held-out tasks (continued).

Task	Core performance	Successful rollouts / 10		
		Qwen3.5 9B base	9B + Complete-task action SFT	**9B + VERA** (k=1)
INV-D05	$V_{\mathrm{OH}} \geq 1.5$ V; $V_{\mathrm{OL}} \leq 0.3$ V; $\mathrm{NM_L}, \mathrm{NM_H} \geq 0.80$ V; $t_{\mathrm{pd,max}} \leq 0.25$ ns	0/10	0/10	**4/10**
INV-D06	$V_{\mathrm{OH}} \geq 1.5$ V; $V_{\mathrm{OL}} \leq 0.3$ V; $\mathrm{NM_L}, \mathrm{NM_H} \geq 0.84$ V; $t_{\mathrm{pd,max}} \leq 0.35$ ns	0/10	0/10	**1/10**
CMP-D05	$V_{\mathrm{OH}} \geq 1.20$ V; $V_{\mathrm{OL}} \leq 0.60$ V; $t_{\mathrm{decision}} \leq 8$ ns; $P \leq 0.5$ mW	0/10	0/10	**1/10**
CMP-D06	$V_{\mathrm{OH}} \geq 1.65$ V; $V_{\mathrm{OL}} \leq 0.15$ V; $t_{\mathrm{decision}} \leq 25$ ns; $P \leq 0.10$ mW	0/10	0/10	**2/10**
Easy total		0/60	0/60	**43/60**
Medium total		0/60	1/60	**22/60**
Hard total		0/60	0/60	**12/60**
Overall		**0/180**	**1/180**	**77/180**

Note: GBW: gain-bandwidth product; SR: slew rate; PM: phase margin; $V_{\mathrm{OH}}/V_{\mathrm{OL}}$: high/low output voltage; $\mathrm{NM_H}/\mathrm{NM_L}$: high/low noise margin; $t_{\mathrm{pd,max}}$: maximum propagation delay; t_{decision}: decision delay; P: power.

B Task Construction and Learning Procedure

The task constructor uses a fixed operator library and seeded sampler to instantiate recovery tasks from verified seeds. Recorded restoration replays certify the generated tasks.

B.1 Target Search and Stopping Rules

Target expansion starts from a verified seed circuit and tightens one performance requirement while retaining the other constraints. For example, it can raise the required gain or lower the permitted power. The new requirement must exceed what the starting seed already satisfies, accounting for measurement tolerance. The learner edits that seed, and a candidate becomes a new seed only after passing the complete specification under the fixed SPICE verifier.

The search visits metrics in a fixed order within their permitted ranges. Each metric has a base step in its physical units. The step multiplier starts at 1, doubles after verified success, and halves after unsuccessful trials across the available metrics. A failed minimum-step search or an exhausted budget stops expansion from that seed. Unresolved targets can be retried after a model update.

A target-search attempt is a learner rollout for a proposed stronger specification; the search budget limits these attempts, while the interaction budget limits edits and evaluations within each rollout. Construction cost includes target-search attempts, certification replay, and failed simulations; learner collection and training are accounted for separately. An unsuccessful search leaves the existing seed available for later target search.

Reference implementation settings. Table 8 lists the reference search configuration shipped in `configs/campaign-all.json`. The released implementation uses a step multiplier in $[0.25, 4]$ and rounds each proposed threshold to the indicated precision. Acceptance tolerances are configured separately and are zero in this reference configuration.

Table 8: Reference search configuration in the released code. Metrics are polled in the listed order within each family.

Family	Metric and direction	Base step	Precision	Threshold range
OTA	Gain lower bound ↑	2 dB	0.1 dB	20–120 dB
	GBW lower bound ↑	0.05 MHz	0.001 MHz	0.01–20 MHz
INV	Delay upper bound ↓	0.1 ns	0.01 ns	0.01–10 ns
	MOS area upper bound ↓	0.1 μm^2	0.01 μm^2	0.1–10 μm^2
CMP	Delay upper bound ↓	1 ns	0.1 ns	0.1–100 ns
	Power upper bound ↓	0.01 mW	0.001 mW	0.001–10 mW

In `vera/campaign.py`, campaign-level limits bound target proposals, accepted expansions, bank size, and model, tool, and evaluation calls. These counters persist across weight updates. The search state resumes with its metric cursor and step size while allowing attempts under the new checkpoint. Task-level budgets separately bound each rollout.

Proposition 1 (accepted escalation). Let $V_{\mathcal{S}}$ be the fixed SPICE verifier for specification $\mathcal{S}$ and $F(\mathcal{S}) = \{C : V_{\mathcal{S}}(C) = 1\}$. If a proposal only tightens existing bounds, the starting seed fails it, and the accepted new circuit passes it, then

$$\varnothing \neq F(\mathcal{S}_{i+1}) \subsetneq F(\mathcal{S}_i). \tag{2}$$

Thus, every accepted target has a verified solution and strictly strengthens its parent's specification. The statement concerns specifications; model difficulty also depends on the initial state and budget.

Proof of Proposition 1. Tightening bounds while retaining all other requirements gives $F(\mathcal{S}_{i+1}) \subseteq F(\mathcal{S}_i)$. The starting seed establishes strict inclusion, and the accepted new circuit establishes nonemptiness under the fixed verifier and its tolerances.

B.2 Recovery-Task Construction and Certification

The constructor removes structure, changes connections, perturbs parameters, or combines these operations on a verified seed. The starting circuit must fail its assigned goal. A private recovery replay must then reach a passing circuit through the learner's editing interface, satisfying the applicable guards and interaction budget. Combined transformations are checked together. Parameter-optimization tasks start from functioning circuits with a performance shortfall; repair tasks may start from nonfunctional circuits with diagnostic feedback.

The learner receives the starting netlist, requirements, permitted edits, budget, and initial SPICE feedback. The seed solution, perturbation labels, and recovery actions remain private. The learner may reach any circuit that passes the assigned task. Identical tasks are deduplicated.

Bootstrap collection uses the fixed 78-task bank with six rollouts per task and no adaptive task renewal. After bootstrap, the reference adaptation rule uses the outcomes of six completed attempts on a recovery task. If at most one attempt succeeds, a multi-operation recipe is split into two shorter recipes, or a parameter perturbation is reduced toward its original value. Each derived task receives a new recovery check. The reference configuration permits up to three combined edits and three levels of recursive task reduction. The construction and scheduling rules are implemented in `vera/learning.py`, `vera/circuit.py`, and `vera/campaign.py`.

Proposition 2 (certified recovery). Suppose replay produces $C_0 \xrightarrow{a_1} \cdots \xrightarrow{a_L} C_L$, every action is permitted, all path guards hold, endpoint guards and the task acceptance condition hold at C_L, and the replay cost fits the declared action and evaluation budgets. Then the derived task has a feasible recovery path within those budgets. This is a constructive existence result; it does not bound the learner's discovery effort.

Proof of Proposition 2. The replay exhibits a legal path from the public starting state to an accepted endpoint, satisfying the guards and budgets.

B.3 Action Supervision and Training Procedure

Section 3.3 defines the action-selection conditions and action-token supervision. Figure 3 illustrates the selected actions.

Bootstrap and recursive updates. Algorithm 1 summarizes bootstrap and recursive training. The seed archive W provides starting designs for target search; only seeds W_k verified under the current round's expanded targets supply its new recovery tasks. The early-expansion condition in Appendix E.2 is a separate first-update comparison.

Algorithm 1 Bootstrap and recursive training in VERA

Input: initial model θ_0, verified seeds W, construction rules, fixed verifier, budgets, and number of updates K
for $k = 0, \ldots, K-1$ **do**
  Keep θ_k fixed during search and collection
  if $k = 0$ **then**
    Construct the fixed certified recovery bank $\mathcal{Q}_0$ from W
    Collect learner trajectories $\mathcal{T}_0$ on $\mathcal{Q}_0$
  else
    Search from W for stronger specifications with θ_k; set W_k to this round's verified seeds and add them to W
    Initialize empty task bank $\mathcal{Q}_k \leftarrow \emptyset$ and trajectory set $\mathcal{T}_k \leftarrow \emptyset$
    while eligible tasks and construction and collection budgets remain **do**
      Construct a certified recovery task from W_k under the expanded targets; skip uncertified tasks
      Record the task in $\mathcal{Q}_k$ and collect learner trajectories into $\mathcal{T}_k$
      Use the outcomes to adjust subsequent practice as described above
    end while
  end if
  Form $\mathcal{D}_k$ from selected actions in successful trajectories in $\mathcal{T}_k$ only
  Partition $\mathcal{D}_k$ into training and validation samples (sample-level random 90:10 split)
  if $\mathcal{D}_k^{\text{train}} = \emptyset$ **then**
    return current model and seed bank
  end if
  Train θ_{k+1} from θ_k on $\mathcal{D}_k^{\text{train}}$ and load the new weights
end for
return updated model and seed bank

Termination. Finite search, task-construction, collection, and training budgets ensure that each round terminates, including when no new target is solved.

C Circuit Subtask Examples

The four circuit task types use certified recovery to expose structural completion, functional repair, metric improvement, and combinations of these objectives. Each task supplies an initial netlist, legal edit scope, guards, and an acceptance predicate. Figure 7 shows their circuit realizations; the shared certification rule is described in Section B.

Construction / modification. A partial circuit requires additions or changes to stages, branches, or connections. For example, removing an OTA's second gain stage leaves the model to construct and connect a stage that meets the assigned gain target.

Diagnosis / repair. A faulty state requires diagnosis from simulator feedback and correction. Perturbing an OTA bias connection can produce a rail-stuck output. The model must restore the response and check the assigned constraints before accepting the candidate.

Parameter optimization. A functioning topology with insufficient performance requires changes to device dimensions, bias, or compensation. For example, perturbed transistor dimensions can reduce OTA gain; the model adjusts them while preserving bandwidth and stability requirements.

Multi-step design. A partial OTA may require adding a gain stage, repairing its bias conditions, and tuning dimensions and compensation. The final candidate must satisfy all assigned constraints together.

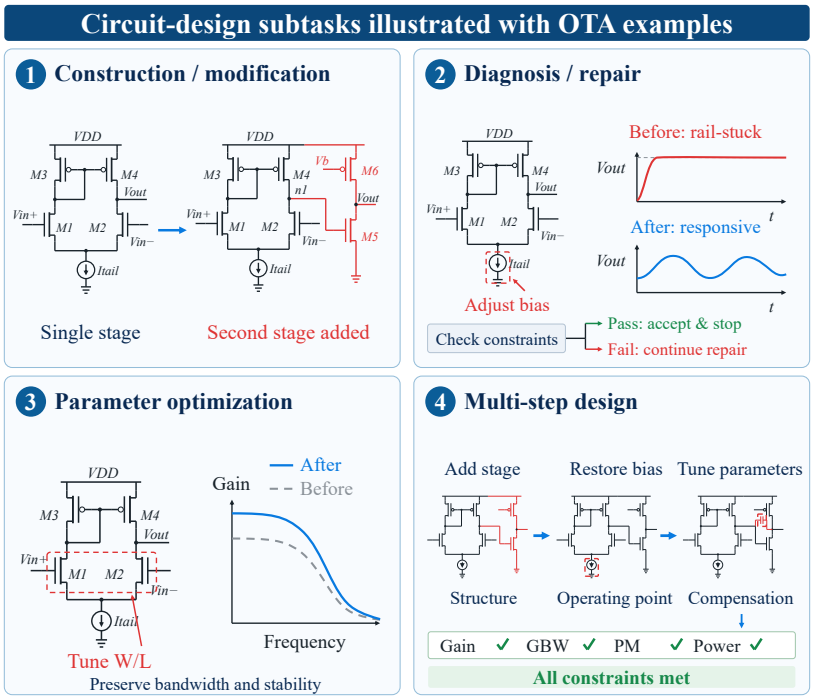


Figure 7: Four recovery-task patterns illustrated with OTA examples.

D External Evaluation Protocol

The 9B rows in Table 9 use Qwen3.5-9B, either as the base model or with the indicated SFT adapter. Qwen3.5-27B is a separate model comparison. GPT-5.6 Terra and Grok-4.6 are evaluated on the 18 internal complete-design tasks in Figure 4.

AnalogCoder-Pro task availability. We evaluate the 20 AnalogCoder-Pro (Lai et al., 2026) task IDs with complete executable evaluation assets (1–8 and 10–21): eight easy, five medium, and seven hard. The remaining tasks lack usable prompt templates, verifiers, or optimization dependencies in the evaluated package. All compared models use this subset.

PhyChip-v2 (Govindugari, n.d.) evaluation covers 50 tasks (L1/L2/L3: 28/15/7). Three generations per task give denominators of 84/45/21 for PhyChip-v2 and 24/15/21 for AnalogCoder-Pro (Table 9). L1 uses the official SPICE execution gate; L2/L3 compare candidate figures of merit against measured reference values under the benchmark harness. Within-group Base/adapter comparisons share tasks, simulator/grader, prompts, and interaction budgets. Reference solutions are excluded from training.

Table 9: External benchmark results by difficulty (successful/evaluated generations). SFT-adapted rows report the first update ($k = 1$). PhyChip-v2 L1 checks execution; L2/L3 use reference-based grading. AnalogCoder-Pro results cover the 20-task executable subset described above.

	PhyChip-v2			AnalogCoder-Pro		
Model	L1	L2	L3	Easy	Medium	Hard
9B base	8/84	0/45	0/21	18/24	4/15	8/21
Qwen3.5-27B	25/84	4/45	0/21	21/24	6/15	11/21
9B + Complete-task action SFT	35/84	0/45	0/21	20/24	7/15	10/21
9B + VERA	**72/84**	**9/45**	**2/21**	**22/24**	**10/15**	**15/21**

E Additional Experiments

E.1 Data Scaling and Collection Controls

Let N_f and N_s denote usable unique action-pool sizes from complete-task and recovery-task sources after quality control and action deduplication. The equal-amount comparison uses $M = \min(N_f, N_s)$.

Data scaling uses nested subsets containing 12.5%, 25%, 50%, and 100% of the $N_{\text{train}} = 1{,}080$-sample bootstrap training partition, stratified by circuit family and task type, with the validation set fixed. Each run starts from the base checkpoint and makes one update. The 100% row corresponds to the first-update checkpoint in the main text.

Table 10: Data scaling over the 1,080-sample training partition. Entries report successful/total rollouts.

Training fraction	Training subset	Internal SR (easy) ↑	PhyChip-v2 / AnalogCoder-Pro
12.5%	135	8/60	19/150 / 26/60
25%	270	17/60	38/150 / 29/60
50%	540	31/60	61/150 / 34/60
100%	1,080	**43/60**	**83/150 / 47/60**

The equal-amount comparison uses a pre-split pool of $M = 160$ action samples from each source. The shared 90:10 split gives 144 training samples and 16 validation samples per source. The collection-budget comparison allocates 90 learner rollouts per source, with each resulting action pool split according to the same protocol (Appendix A.3).

Table 11: Equal-data and equal-collection-rollout controls.

Comparison	Training source	Action pool size	Learner rollouts	SR (easy) ↑
Equal amount	Complete-task actions	$M = 160$	90	0/60
	Recovery-task actions	$M = 160$	∼62	9/60
Equal collection-rollout budget	Complete-task collection	∼160	90	0/60
	Recovery-task collection (VERA)	∼230	90	12/60

These controls show higher downstream SR for recovery-task supervision at equal action amount (9/60 versus 0/60) and at equal learner-collection rollout budget (12/60 versus 0/60).

E.2 Task Construction for Bootstrapping

Table 4 compares target expansion before the first update (V) and recovery-task construction (H). With H off, collection uses complete-design tasks; with H on, it uses certified recovery tasks. With

V on, stronger-target search precedes training. Target search starts from a verified seed; collection receives the starting circuit and public specification of its assigned complete-design or recovery task. Constructor recovery records remain private. Each pipeline uses the shared protocol in Appendix A.3 with its own action pool.

Off/Off reuses the Complete-task action SFT experiment in Table 1, including usable local actions from failed episodes (Appendix A). Off/On reuses the bootstrap corpus and first-update checkpoint. These rows compare the pipelines with their actual data yields and collection counts; Appendix E.1 provides equal-amount and equal-collection-budget controls. On/On adds expansion before the initial recovery-task training step; Appendix E.4 evaluates subsequent recursive updates.

E.3 Supervision Ablations

This study compares supervision variants on the same raw recovery trajectories, with each rule yielding its own action-pool size. The **full method** retains Pass/Repair/Progress actions from episodes whose final candidate passes the task. The **successful-trajectory** control uses the same successful episodes but supervises all assistant actions. The **no-endpoint-gate** control applies the action rules without requiring final success. The full-method and successful-trajectory rows correspond to their first-update checkpoints in Table 1.

Table 12: Supervision variants on the same recovery-trajectory pool.

Variant	Action pool	SR (easy) ↑
Full VERA selection	1,200	**43/60**
All actions in successful trajectories	3,850	27/60
Selected actions without endpoint gate	1,460	31/60

E.4 Recursive Task-Supply Comparisons

All branches continue from the common first-update checkpoint θ_1. The full-recursion results correspond to the $k = 2$ and $k = 3$ rows of Table 1.

Section 4.5 and Table 3 define the search model, task bank, and collection learner for each branch. Table 14 reports each update's selected-action pool and cumulative learner collection.

Table 13: Recursive task-renewal controls. All branches start from the same θ_1; results cover all 18 complete-design tasks.

Condition	Data for next update	θ_2 SR ↑	θ_3 SR ↑
Fixed bank	Fresh rollouts on fixed Q_0	91/180	97/180
Full recursion	Current checkpoint on new Q_k	**109/180**	**123/180**
Frozen search model	Current learner on new Q_k^{fw}	95/180	103/180

Table 14: Task supply and learner collection across recursive updates.

Checkpoint / condition	SR ↑	New coverage	Selected pool size	Cumulative learner rollouts
θ_0	0/180	n/a	n/a	0
θ_1	77/180	n/a	1,200	468
θ_2, fixed bank	91/180	n/a	1,350	936
θ_2, full recursion	**109/180**	+14 targets / +9 seeds	1,480	1,026
θ_2, frozen search	95/180	+6 targets / +3 seeds	1,340	960
θ_3, fixed bank	97/180	n/a	1,480	1,404
θ_3, full recursion	**123/180**	+11 targets / +7 seeds	1,650	1,566
θ_3, frozen search	103/180	+5 targets / +2 seeds	1,410	1,446

Note: New coverage reports separate counts of target specifications and accepted seed circuits added for that update; subtasks are certified recovery exercises constructed from seeds. Cumulative rollouts count learner collection, excluding construction and certification. Full recursion and frozen search collect on recovery tasks constructed under each round's expanded targets. The fixed-bank control alone continues collection on Q_0.

F Applicability beyond Circuit Design

Extending VERA requires verified seed artifacts, editable states, executable acceptance checks, and an ordering of stronger targets that retains prior constraints. Task transformations must provide legal recovery paths through the learner's action interface within the declared budgets. Applications beyond circuits require domain-specific transformations, verifiers, and empirical evaluation.