

ANALOG DESIGN AUTOMATION

Fine-tuning Qwen3.6-27B with trace data

A two-stage SFT+GRPO pipeline and **hybrid deployment** for analog circuit optimization

AUTHOR

Neurologic AI

PUBLISHED

July 2026

SIMULATOR

ngspice

ABSTRACT

A companion paper described a system for collecting structured, replayable traces of expert-style reasoning about analog circuit sizing: a large reasoning model iteratively proposing, simulating, and correcting parameter changes against a fixed circuit topology. This paper describes what happens to that trace data next. We convert collected traces into two distinct training corpora from a single shared prompt builder, an action-only supervised fine-tuning (SFT) corpus that imitates a teacher model to install a correct output format and reasoning style, and a set of reinforcement-learning seeds used to train beyond that teacher

with Group Relative Policy Optimization (GRPO), using the same circuit simulator that generated the original traces as a ground-truth reward. We fine-tune Qwen3.6-27B with this pipeline and identify and correct a subtle reward-hacking vulnerability in the naive reward function, in which a circuit measurement that silently fails to simulate is scored better than one that is measured and found wanting, an incentive that would otherwise teach the policy to break the simulator rather than meet the design specification. At deployment, the fine-tuned model is paired with a proprietary, more capable model in a two-phase hybrid architecture: the fine-tuned model runs first and for free, and the proprietary model is escalated only on the subset of circuits where the fine-tuned model's trajectory shows clear evidence of stalling, resuming from its best-found state rather than restarting. On a controlled ten-circuit benchmark spanning four LDO regulators and six operational amplifiers, this hybrid system matched or exceeded both single-model baselines on nine of ten circuits, fully satisfied every design specification on three circuits where neither baseline did so alone, and reduced proprietary-model token consumption by approximately 72% relative to running the proprietary model alone. These results indicate that a domain-specialized, RL-trained small model, combined with a disciplined and evidence-based escalation policy rather than a fixed model choice, can recover most of the quality of a much larger general-purpose model at a fraction of its cost.

9 of 10

circuits where the hybrid matched or exceeded both single-model baselines

3

circuits fully satisfied where neither baseline did so alone

≈72%

fewer proprietary-model tokens than running the proprietary model alone

Introduction

Once a collection system has produced a large set of structured reasoning traces for analog circuit sizing, as described in a companion paper, a natural question follows: how should that data be used to produce a working, deployable optimization agent, and does the resulting agent need to be the same large, expensive model that generated the traces in the first place? The traces themselves are demonstrations of a capable teacher model reasoning its way through a design problem; simply imitating them, in the manner of standard supervised fine-tuning, would at best reproduce the teacher's behavior and would inherit any of its blind spots or inefficiencies without correction. At the same time, discarding the teacher's demonstrations entirely and training a policy from reinforcement learning alone would waste a large amount of informative, real signal about which sequences of reasoning and action actually lead to a converged circuit, and would likely be far less sample-efficient in a domain where every training signal requires an expensive circuit simulation to obtain.

This paper adopts a two-stage answer to that question, following a pattern that has proven effective elsewhere in language-model post-training: a supervised cold start followed by reinforcement learning against a verifiable, ground-truth reward. In the first stage, the fine-tuned model, Qwen3.6-27B, is trained to imitate a reduced form of the teacher's reasoning and output format, establishing a correct, parseable action schema before any exploration is asked of it. In the second stage, Group Relative Policy Optimization is used to let the model improve beyond its teacher, using repeated circuit simulation in ngspice as a ground-truth reward signal rather than a learned or model-based judge. Because circuit simulation is authoritative, deterministic, and directly tied to the actual design specification, it offers a considerably more trustworthy reward than an approximate or learned alternative, provided the reward function built on top of it is designed carefully enough to resist being gamed. A substantial part of this paper is devoted to exactly that design problem: a naive reward built on the same scoring function used to grade circuits during trace collection contains a specific vulnerability, in which a metric that fails to simulate is scored better than one that is measured and found to be failing, creating a direct incentive for the policy to learn to break the simulator rather than solve the circuit. Identifying and closing this vulnerability is treated here as a first-class part of the training methodology, not an afterthought.

A second question follows once a fine-tuned model exists: is it, on its own, a sufficient replacement for the far larger, more expensive proprietary model whose behavior it was trained to imitate and exceed? For at least a subset of circuits, the answer is no; a domain-

specialized model trained primarily on trajectories from a fixed set of circuit families can be expected to underperform a stronger general-purpose reasoning model on cases that fall further from that training distribution. Rather than treating this as a reason to fall back on the expensive model for every circuit, or to accept degraded quality on the hard cases, this paper describes a hybrid deployment architecture in which the fine-tuned model always runs first, and the proprietary model is escalated only when the fine-tuned model's own trajectory provides clear, legible evidence that it has stalled, picking up from exactly where the fine-tuned model left off rather than starting over. This turns the choice of which model should optimize a given circuit from a fixed, a priori decision into a per-run, evidence-based one.

The remainder of this paper is organized as follows. Section 2 describes how the raw trace data from the collection stage is converted into training-ready inputs through a single shared prompt-rendering component, which is what keeps every later stage on-distribution with respect to every earlier one. Section 3 describes the supervised fine-tuning stage in detail, including the reduced output schema, the data-filtering criteria, and a subtle data-quality bug and its fix. Section 4 describes the GRPO reinforcement learning stage, including how training seeds are constructed, how the reward function is designed and hardened against the specific exploit identified above, and the multi-circuit training architecture that lets the same policy be trained jointly across many distinct circuit families. Section 5 describes the hybrid deployment architecture and the escalation policy that governs the hand-off between the fine-tuned model and the proprietary model. Section 6 presents benchmark results comparing the hybrid system against both single-model baselines. Section 7 concludes.

02

From trace to training data

2.1 A shared prompt builder as a binding invariant

A single design decision underlies every stage described in this paper: one shared prompt-rendering component (comprising a system-prompt renderer and a user-message renderer) is used to construct model inputs during trace collection, supervised fine-tuning, GRPO rollout, and eventual deployment alike. This is treated as a binding invariant of the entire pipeline rather than an implementation convenience, because it directly determines whether the different stages remain compatible with one another. If the prompts a model

sees during its supervised cold start differed structurally from the prompts it sees during reinforcement learning rollouts, the policy would effectively be asked to generalize across a distribution shift it was never trained to handle; the same is true, with a correspondingly higher deployment cost, if the prompts seen at inference time differed from either training stage. Rendering every stage from the same builder collapses that risk by construction: whatever format the model has learned to interpret is exactly the format it will continue to see, from cold start through eventual deployment.

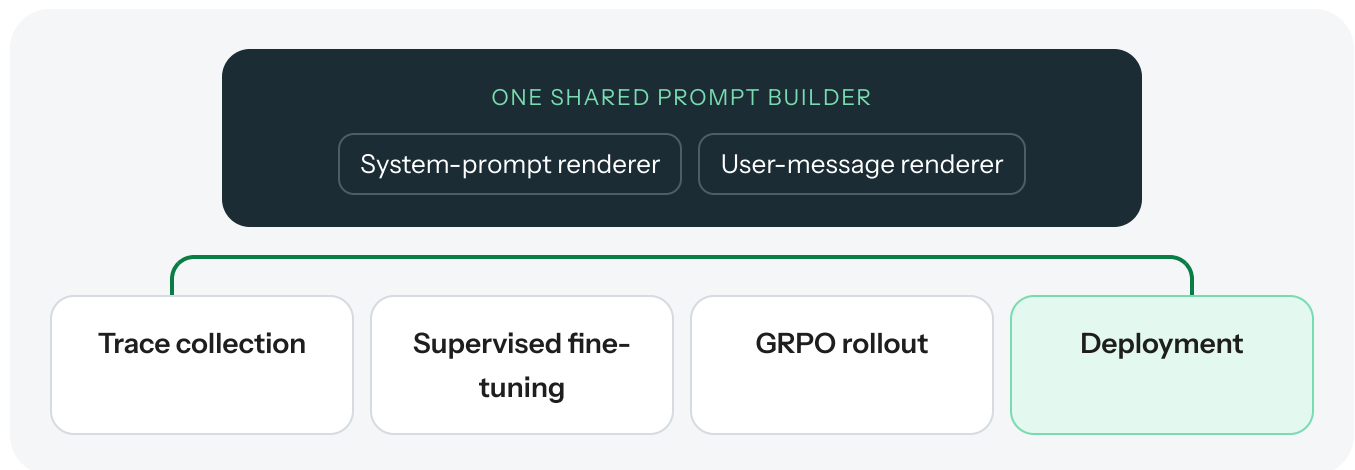


Figure 1. The binding invariant: every stage renders its inputs from the same builder, so the format the model learns is exactly the format it will continue to see.

2.2 Source data: the collection traces

Both the supervised fine-tuning corpus and the GRPO training seeds are derived from the same underlying artifact: the per-run optimization traces produced by the collection system, stored as one JSONL file per run. Each trace consists of a header row, capturing the circuit's fixed context (its process-design-kit bins, its knob table, its topology, and its testbench) together with its design targets and circuit identifier; a baseline row, capturing the circuit's starting parameters and initial score; and a sequence of step rows, each recording the parameter state before the step, the per-target score before the step, the action taken, the model's decision about how to proceed, its stated rationale for the action, its updated search strategy, the resulting change in penalty, and the score after the step.

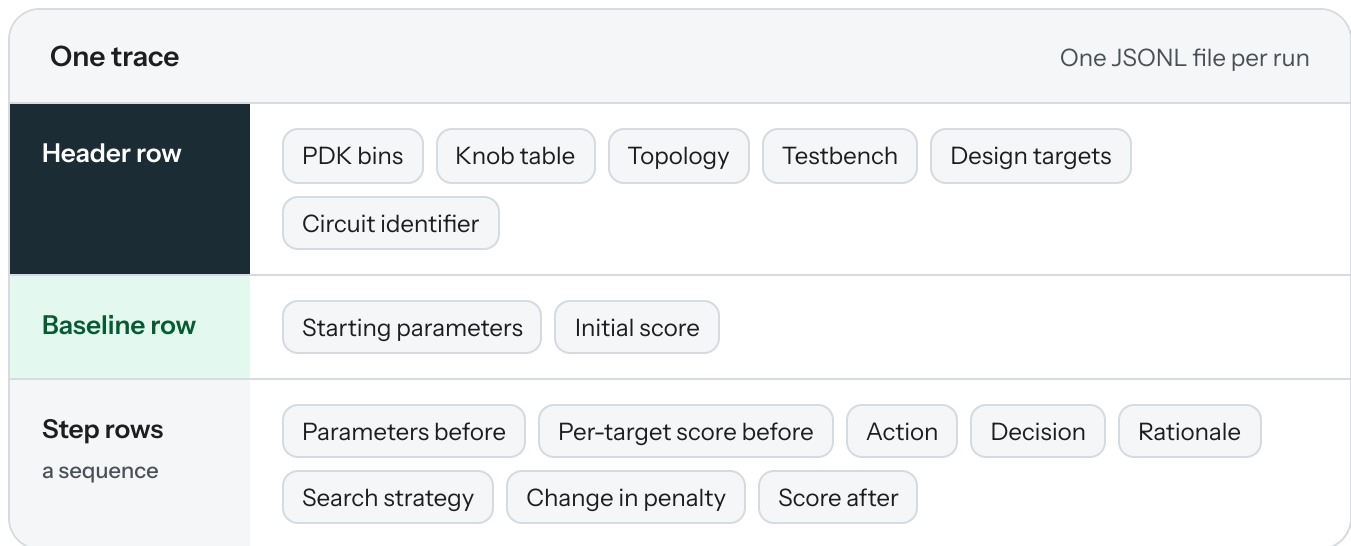


Figure 2. Anatomy of a collection trace. A trace stores raw ingredients rather than a pre-rendered prompt.

A deliberate design choice underlies this format: a trace stores raw ingredients rather than a pre-rendered prompt. The conversion step described in the following sections re-renders the actual model input from these fields at training time, using the same shared builder described above, rather than storing and replaying a frozen prompt captured at collection time. This keeps both downstream training stages synchronized with a single source of truth for prompt structure, so that a later change to prompt formatting need only be made once.

03

Supervised fine-tuning: cold-starting the policy

The purpose of the supervised fine-tuning stage is narrow and specific: install a correct, reliably parseable output format and a reasonable reasoning style before any reinforcement learning is attempted, rather than to teach the model everything it will ultimately need to know about circuit optimization. Each training sample is constructed as a chat triple, a system message, a user message, and an assistant response, together with a metadata block used only for splitting and loss weighting and never shown to the model.

3.1 A reduced output schema

The teacher model's original output format included six distinct fields per turn, but the fine-tuned policy's target schema was deliberately reduced to only what the surrounding environment and reward function actually consume: a free-form reasoning block, followed

by an action expressed as JSON mapping knob names to proposed values (or an empty action to indicate a deliberate no-op). Several of the teacher's original fields were dropped from the label on a consistent principle, that a field which is never read by the reward function or the next turn's prompt is a field that only adds parsing risk without adding learning signal. The teacher's explicit numeric predictions, for instance, were removed because the reward is computed entirely from simulated outcomes and never grades whether a prediction was accurate; a field describing perceived topology limitations was removed because it would otherwise function as a policy-controlled mechanism capable of altering how the circuit is scored, which is precisely the kind of surface a reward-seeking policy could learn to exploit; and an explicit termination decision was removed because run termination is owned by the surrounding loop's own convergence and stagnation logic, not by the policy's self-report. The only field besides the reasoning block that survives into the label is the action itself, since it is the only field actually applied to the simulated circuit and scored. Reducing the number of required output fields correspondingly reduces the number of ways a generated completion can fail to parse, which in turn reduces the variance of the reward signal used in the later reinforcement learning stage.

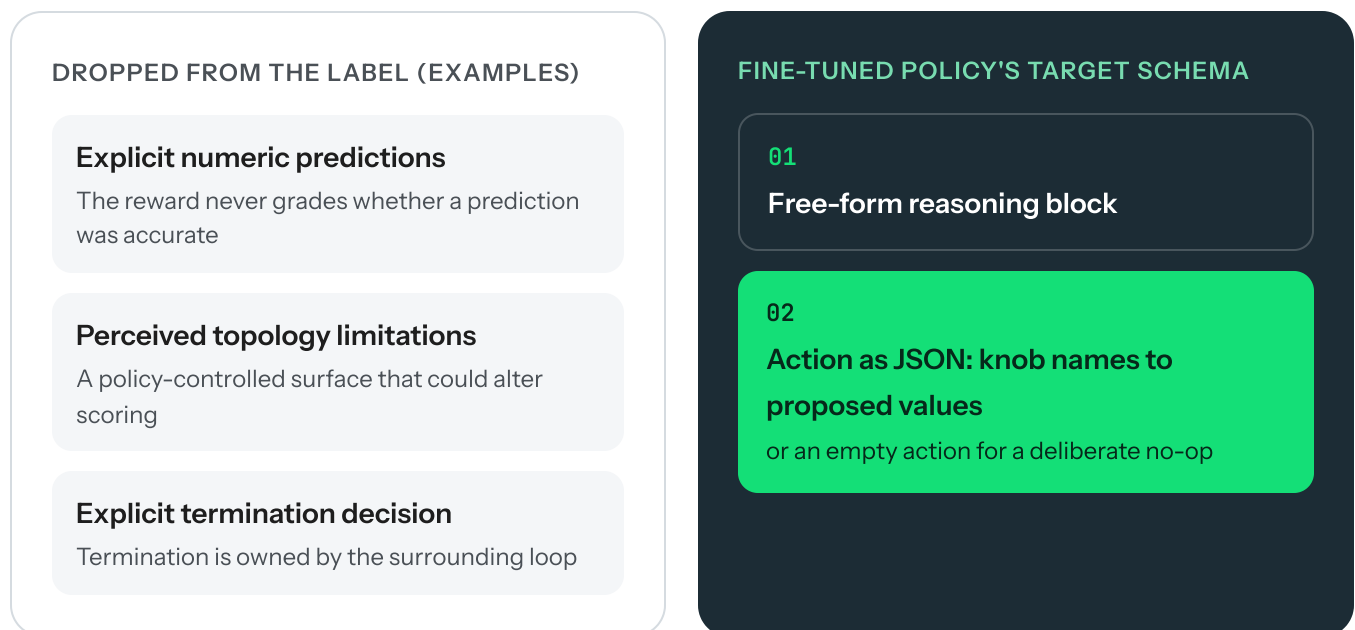


Figure 3. The reduced output schema: a field never read by the reward function or the next turn's prompt only adds parsing risk.

3.2 System and user prompt construction

The system prompt is rendered per circuit from that circuit's stored context, and is deliberately written in process-agnostic language (referring to a "fixed circuit topology" rather than naming a specific device family), so that nothing about the underlying

fabrication process is hard-coded into the prompt template itself; all device-specific information is supplied by the data. The discrete set of legal device geometries is included explicitly in this prompt, so that the validity component of the later reward function never penalizes the policy for violating a constraint it was never actually shown.

The user message is rendered fresh at every step, and includes the circuit's current knob values, a target table reporting each metric's measured value, its remaining gap to target, and its pass/fail status (with an explicit accounting for metrics that failed to be measured at all), a repair-mode warning when a hard-floor metric is in an invalid operating regime, the best state seen so far in the run, and a rolling history of recent steps. That history is not a simple sliding window over the most recent turns; it is the union of the ten most recent steps and the four steps with the largest absolute change in penalty across the entire run so far, rendered in chronological order, so that an important mistake made early in a long run remains visible to the model even after many subsequent turns, rather than silently aging out of context.

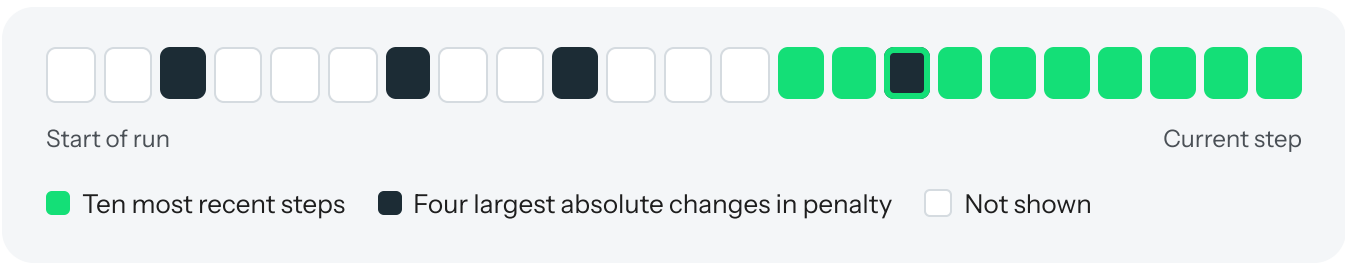


Figure 4. The rolling history, shown on an illustrative run: the union of the ten most recent steps and the four highest-impact steps, rendered in chronological order, so an early mistake stays visible.

3.3 Filtering criteria

Not every step recorded during trace collection is suitable for inclusion in the supervised fine-tuning corpus. The default and primary filter is validity alone: a step is retained if and only if its action was legal at the time it was taken (an on-catalog device geometry, an integer multiplier of at least one, an in-range bias current, applied to a knob the model was actually permitted to touch, and not the result of an internal error fallback). Deliberately, no additional quality gate is applied on top of this: whether a given step happened to improve the circuit's score is not used to decide whether it belongs in the cold-start corpus, since judging and rewarding outcome quality is treated as the explicit job of the later GRPO stage, not the supervised stage. A cold start is intended to expose the policy to broad, valid coverage of the state space it will need to reason about, and gating training examples by outcome quality this early would instead install a greedy, potentially premature prior. Steps

in which the model chose to revert, hold, or stop are retained for the same reason: excluding them would leave the policy with no exposure at all to recovery behavior, which is precisely the behavior needed when an earlier action has made things worse.

THE FILTER: VALIDITY ALONE

On-catalog device geometry

Integer multiplier of at least one

In-range bias current

Permitted knob

Not an internal error fallback

No outcome-quality gate. Revert, hold and stop steps are retained.

3.4 A data-quality fix: hypothesis inlining

A subtle but consequential data-quality issue was identified in how the reasoning block references hypotheses. The reasoning text produced during trace collection frequently refers to a hypothesis by its identifier, for instance stating that a particular hypothesis has now been ruled out, but the actual textual description of that hypothesis lives in a separate, nested field recorded at the step where the hypothesis was first proposed, not at the step where it is later referenced or ruled out. Because a hypothesis can be proposed at one step and rejected several steps later, naively rendering only the current step's data would produce a reasoning block that references a hypothesis identifier whose description the model was never actually shown, an artifact the fine-tuned model would otherwise learn to imitate: confidently citing and dismissing a hypothesis it had never itself stated.

The fix walks a trace's steps in order, accumulating a running registry mapping each hypothesis identifier to its description as hypotheses are introduced, and for each reasoning block, inlines only the descriptions of the specific identifiers that block actually references. This registry is used purely as a lookup source during data construction; its contents are never emitted directly into any training example. Because a run's own set of live hypotheses is self-pruning, typically three to six active hypotheses at any point, this fix does not cause reasoning blocks to grow unboundedly with episode length even though the underlying registry itself continues to grow. Correcting this issue required regenerating the supervised fine-tuning corpus in full and retraining from the corrected data.

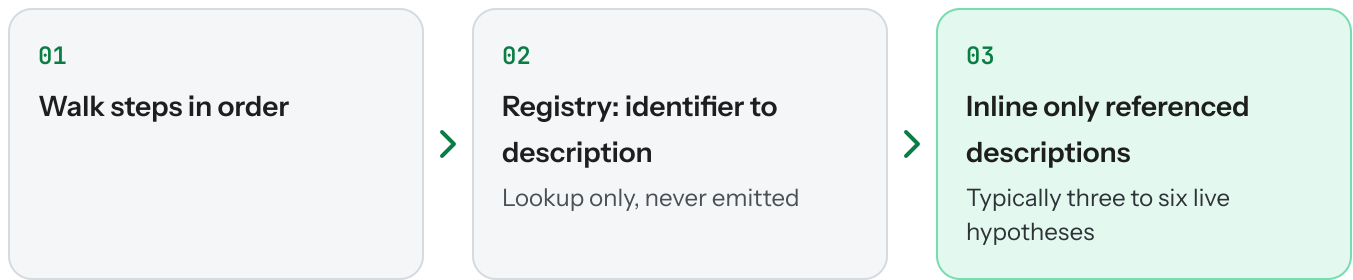
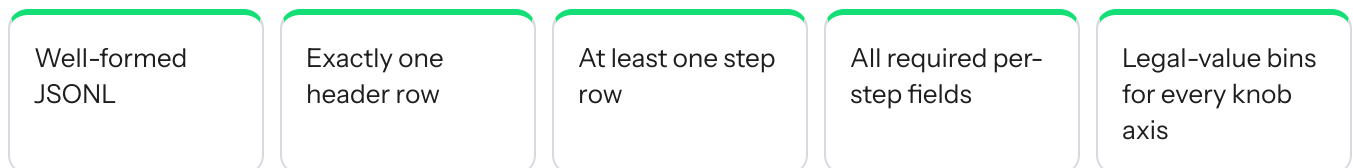


Figure 5. Hypothesis inlining, so the model never cites a hypothesis whose description it was not shown.

3.5 Preflight validation

Before a trace is used to build training examples, it is checked for structural completeness, and any trace found to be malformed is skipped and logged rather than silently included or allowed to halt the entire build. Checks include well-formed JSONL, the presence of exactly one header row, the presence of at least one step row, the presence of all required per-step fields, and the presence of discrete legal-value bins for every relevant knob axis. As an additional consistency check, the builder verifies that the legal-value bins it renders into a prompt match the bins actually recorded in that trace, and separately warns whenever a teacher action falls off those bins, which is treated as a signal that a circuit's stored context may have drifted out of sync with the process design kit it was originally generated against.



3.6 The native-thinking regime

An important and initially counter-intuitive empirical finding concerns how the model's chat template interacts with its own reasoning behavior. The base model's template injects the opening reasoning-block marker into the prompt itself rather than requiring the model to generate it, which means the model was effectively trained to continue an already-open reasoning block, not to open a new one on its own. The practical consequence is that at both inference time and GRPO rollout time, the model must be run with this native-thinking behavior enabled so that the template continues to supply the opening marker; the model's own completions then consist of reasoning text followed by the closing marker and the action, typically without an opening marker of their own. Every downstream component that needs to separate reasoning from action must therefore parse by locating the closing marker rather than the opening one, since the opening marker is essentially never present

in the model's own generated text. This corrected an earlier assumption that training had used the alternative regime, in which the model would be expected to generate its own opening marker.

PARSING RULE

Parse by locating the closing marker rather than the opening one.

3.7 Training configuration

Supervised fine-tuning uses a standard Trainer implementation together with parameter-efficient fine-tuning, with the training loss masked so that it is computed only over the assistant's turn, including its reasoning block, and not over the system or user portions of the prompt. Training and validation splits are assigned at the level of entire runs, keyed by circuit identity, with an explicit leakage guard that warns if the same circuit ever appears on both sides of the split. Training runs under data-parallel replication across two GPUs, since the 27-billion-parameter model fits comfortably on a single high-memory accelerator and is therefore replicated rather than sharded; a particular checkpointing setting (disabling reentrant activation checkpointing) is required under this configuration, since the reentrant variant double-fires the parallel-training hooks attached to the low-rank adaptation parameters otherwise. Samples that exceed the configured maximum sequence length are skipped outright rather than truncated, since truncating a reasoning trace risks silently cutting off the action it is meant to justify. Low-rank adaptation is applied with rank 32 and scaling factor 64 across the attention projections and the gated feed-forward projections; the longest single training sample observed was approximately 9,600 tokens, which set the required maximum sequence length for the run.

LORA RANK

32

SCALING FACTOR

64

LONGEST SAMPLE

≈9,600

DATA-PARALLEL

2 GPUs

Reinforcement learning with GRPO

4.1 From imitation to self-improvement

Where supervised fine-tuning teaches the policy to imitate a fixed set of recorded teacher trajectories, Group Relative Policy Optimization is used to let the policy improve beyond those trajectories by re-solving each circuit itself. Given a starting circuit state, the policy proposes a knob change, the change is simulated in ngspice, a reward function scores the outcome, and a group-relative advantage computed across multiple sampled completions for the same state drives a policy-gradient update, regularized by a Kullback-Leibler penalty back toward the frozen supervised-fine-tuning policy so that the model does not drift arbitrarily far from its cold start. The formulation used for this stage of training is single-step: each individual training example corresponds to one seeded circuit state, several candidate completions are sampled and independently simulated and scored for that state, and the reward combines the immediate change in the state's potential with a bonus for reaching a fully converged circuit. This is considerably simpler and lower-variance than a full multi-step episodic formulation, and is treated as the appropriate way to first establish that the reinforcement learning loop itself is stable, before adding the additional complexity of trajectory-level credit assignment.

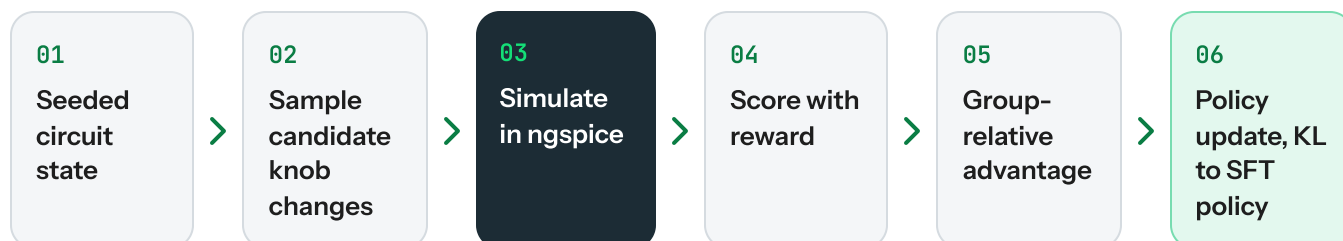


Figure 6. The single-step GRPO formulation: one seeded state per training example, several candidate completions simulated and scored.

4.2 Constructing GRPO seeds

A GRPO training seed is a prompt-only entry, a system message and a user message with no accompanying assistant turn, since the policy is expected to generate its own action rather than imitate a recorded one; critically, a seed represents a starting state to roll out from, not a trajectory to imitate.

4.2.1 Baseline: one seed per circuit

The simplest seed-construction scheme, and the default behavior of the data-conversion tool, emits exactly one seed per collected trace, corresponding to that trace's starting baseline state; a sixteen-step trace under this scheme yields exactly one GRPO seed rather than sixteen, in contrast with the supervised fine-tuning corpus, where the same sixteen-step trace yields approximately sixteen individual training samples, one per step. Because the policy generates its own subsequent steps from a seed via live simulation, the specific steps the teacher happened to take from that same starting point are irrelevant to the seed itself; only the state the episode begins from matters. This baseline scheme is clean and easy to reason about, but, as discussed in Section 4.6, it has a real limitation for a policy meant to run many iterations at deployment.

4.2.2 Intermediate seeds: covering the state space

To address that limitation directly, the seed-construction tool can instead emit multiple seeds per circuit, sampled from different stages of the teacher's original trajectory: for a configured number of seeds per circuit, it selects that many step-states, evenly spaced across the run, so that the policy is exposed to baseline, early, middle, and late states from the same underlying trajectory. A configuration selecting five seeds from a sixteen-step trace, for instance, yields seeds drawn from approximately steps one, five, nine, twelve, and sixteen. Each intermediate seed genuinely represents a different, partially optimized circuit state, since it uses that step's parameters as they existed after the teacher's preceding moves had already been applied; empirically, intermediate seeds were confirmed to differ from their trace's baseline state in the large majority of knob values. States that have already converged are deliberately excluded from this scheme, since there is nothing left to optimize from an already-solved circuit; convergence for this purpose is judged using the same corrected, reward-side convergence definition described in Section 4.3.2, so that a state which only appears to pass because a metric silently failed to measure is not mistaken for a genuinely solved seed, while states that are close to converged but still have a small number of failing metrics are retained as legitimately useful, harder training seeds. Because the number of seeds drawn is set per circuit rather than in proportion to trace length, long traces do not disproportionately dominate the resulting seed set relative to short ones; across the full set of roughly ninety circuits, drawing approximately eight seeds per circuit yields several hundred seeds, providing meaningfully broader state-space coverage without requiring any additional simulation beyond what trace collection already produced.

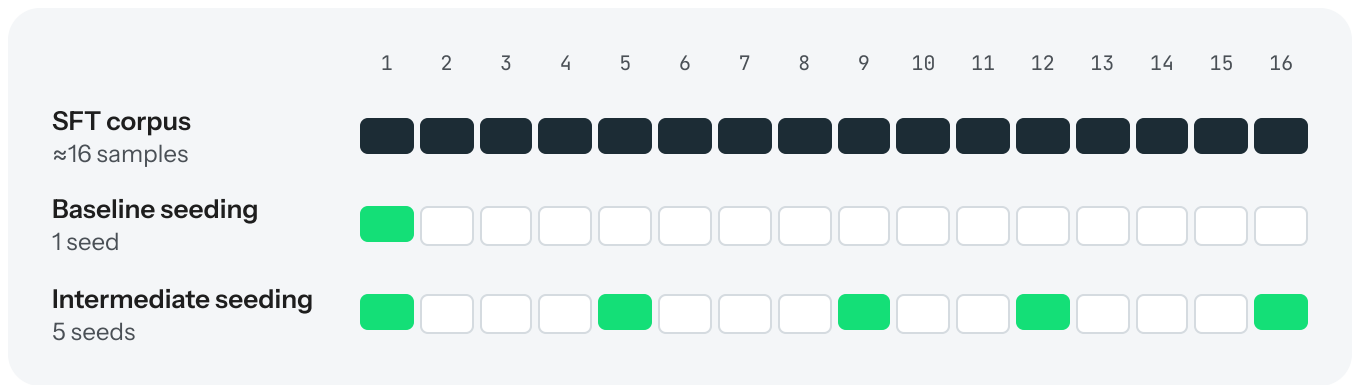


Figure 7. What one sixteen-step trace yields: about sixteen SFT samples, one baseline seed, or five intermediate seeds drawn from approximately steps one, five, nine, twelve and sixteen.

4.3 Reward design

4.3.1 A tiered, potential-based reward

The reward used for GRPO is built directly on top of the same penalty-scoring function used to grade circuits during trace collection, verified to be smooth and bounded and normalized per metric, which makes it suitable to serve as a potential function for reinforcement learning. The reward itself is organized into three tiers. The first tier consists of hard gates applied before any simulation occurs: a completion that fails to parse, or whose proposed action is invalid (an off-catalog device geometry, a non-integer multiplier, an out-of-range bias current, or an attempt to touch a knob the policy is not permitted to touch), receives a fixed negative reward and the step ends immediately without ever reaching the simulator. The second tier is a dense, potential-based progress reward, proportional to the reduction in total penalty achieved by the step (the negative of penalty serving as the potential function) minus a small fixed per-step cost; this shaping is dense, cannot be exploited by oscillating back and forth between two states, and does not reward continuing to improve a target that has already passed its threshold. The third tier is a terminal bonus awarded when the circuit reaches full convergence according to the scorer's own pass/fail determination, again combined with a small per-step cost so that faster convergence is scored more favorably than slower convergence to the same final result. Because the hard-floor penalty applied to genuinely oscillating circuits is in principle unbounded, the resulting reward is clipped at a fixed magnitude, so that a single catastrophic rollout cannot dominate the group-relative advantage normalization computed across an entire batch. Consistent with the overall design, episode termination is owned entirely by the surrounding loop's convergence and stagnation logic; the policy itself never emits an explicit stop signal that the reward needs to interpret.

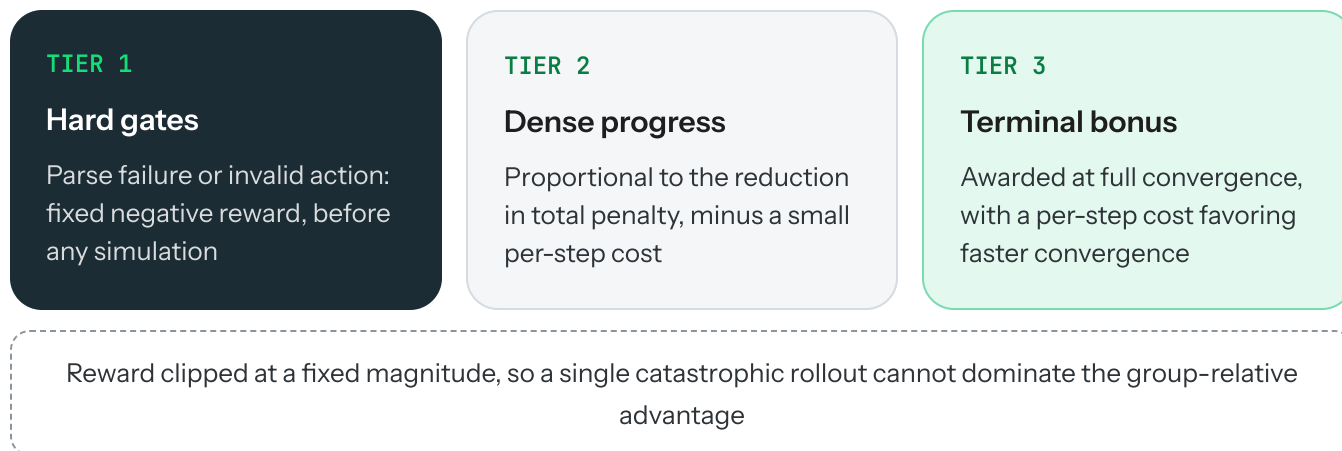


Figure 8. The three reward tiers.

4.3.2 The neutral-miss reward-hack fix

The single most important correctness property of the entire reward design concerns how it handles a metric that fails to simulate at all, whether due to a simulation error, a metric that was never emitted, or an extraction failure during post-processing. The original scoring function, reasonably enough for its original purpose of grading a trusted collection process against occasional tooling glitches, treats such a metric charitably: it assigns a low penalty and marks the metric as passing. Under reinforcement learning, this charitable treatment becomes a direct exploit: an action that causes a metric to fail to be measured at all scores substantially better, under the original scoring function, than an action that causes the metric to be measured and found to be failing, and in some cases a missing metric can even trigger the scorer's own convergence flag. Left uncorrected, the policy would have every incentive to learn to break the simulator rather than to actually meet the design specification, and, worse, the training run would appear to be succeeding, with both reward and apparent convergence rising, while producing circuits that do not actually work.

The fix is applied entirely on the reward side, without altering the original scoring function used elsewhere, and recomputes penalty and convergence so that any metric with no measured value is treated as a failing metric rather than a passing one: a circuit-induced measurement miss is assigned a failing contribution deliberately set above the penalty ceiling for a metric that was measured and found to be failing, and is explicitly excluded from the convergence determination, so that an unmeasured metric can never be mistaken for a passing one and can never, on its own, satisfy the terminal convergence bonus; separately, a complete simulation failure, in which every metric comes back unmeasured, triggers the same hard gate used for invalid actions and ends the step immediately. Verification against the corrected reward confirms the intended ordering: a case that the original scorer would grade as a low penalty with convergence satisfied is correctly reassigned a substantially

higher penalty with convergence denied, placing "broken measurement" strictly below "measured and failing" in preference, exactly the ordering needed to remove any incentive to break the simulator.

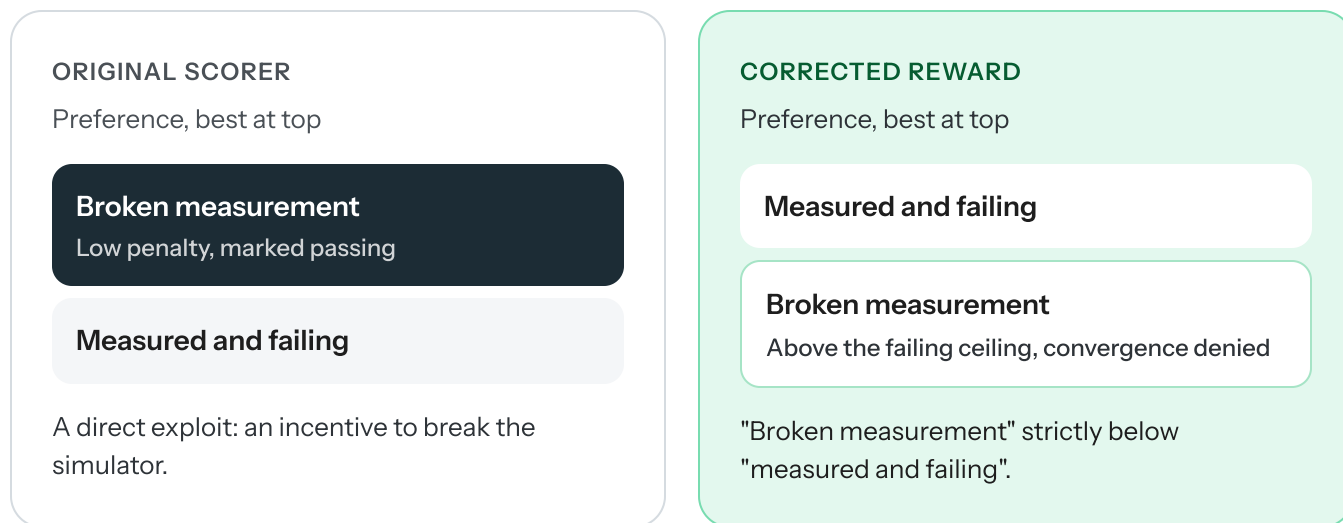


Figure 9. The neutral-miss fix reorders preference so that an unmeasured metric can never be mistaken for a passing one.

4.3.3 Why not an LLM judge

An alternative design would use a separately trained or prompted language model as a judge to score circuit behavior, rather than relying on a physics-based circuit simulator. This alternative is deliberately rejected as the training signal here: ngspice functions as a ground-truth oracle for circuit behavior, and a judge model could, at best, only approximate the same underlying numbers, with additional noise and no additional information, while simultaneously reopening exactly the kind of reward-hacking surface that Section 4.3.2 was written to close. A learned judge's legitimate role in this pipeline is limited to offline supervised-data curation and post hoc evaluation; it is deliberately excluded from serving as the reinforcement learning training signal itself.

4.4 Multi-circuit training architecture

Because every circuit carries its own design targets, its own set of adjustable knobs, its own testbench, and its own supply conditions, both the scoring function and the simulator used during GRPO training must be resolved per circuit rather than shared globally. This is handled through a registry, built once at startup, that scans a directory of circuit definitions and constructs, for each circuit, the specific scorer and simulation routine appropriate to it, together with its knob context and supply voltage; separate registries are built from the training and validation circuit directories, so that the training update and the validation

evaluation always draw from disjoint circuit sets. During training, the reward function resolves each sampled completion's originating circuit from its associated metadata, looks up the corresponding entry in the training registry, and scores the completion's action against that specific circuit's simulator and target specification, falling back to a single shared circuit configuration only in the degenerate case where just one circuit is being trained on at all. At the simulation level itself, a proposed set of knob values is injected into a pre-rendered, per-circuit testbench template, copied into an isolated working directory so that concurrent rollouts do not interfere with one another or mutate any shared source files, and the resulting ngspice run is parsed and post-processed using the same metric-derivation logic used during the original trace collection, so that a metric computed during GRPO training is guaranteed to be defined identically to the same metric computed during data collection. As in the reward design above, any metric absent from this parse is treated as a missing value and handled by the same neutral-miss override described in Section 4.3.2, and a complete simulation failure triggers the same hard gate.

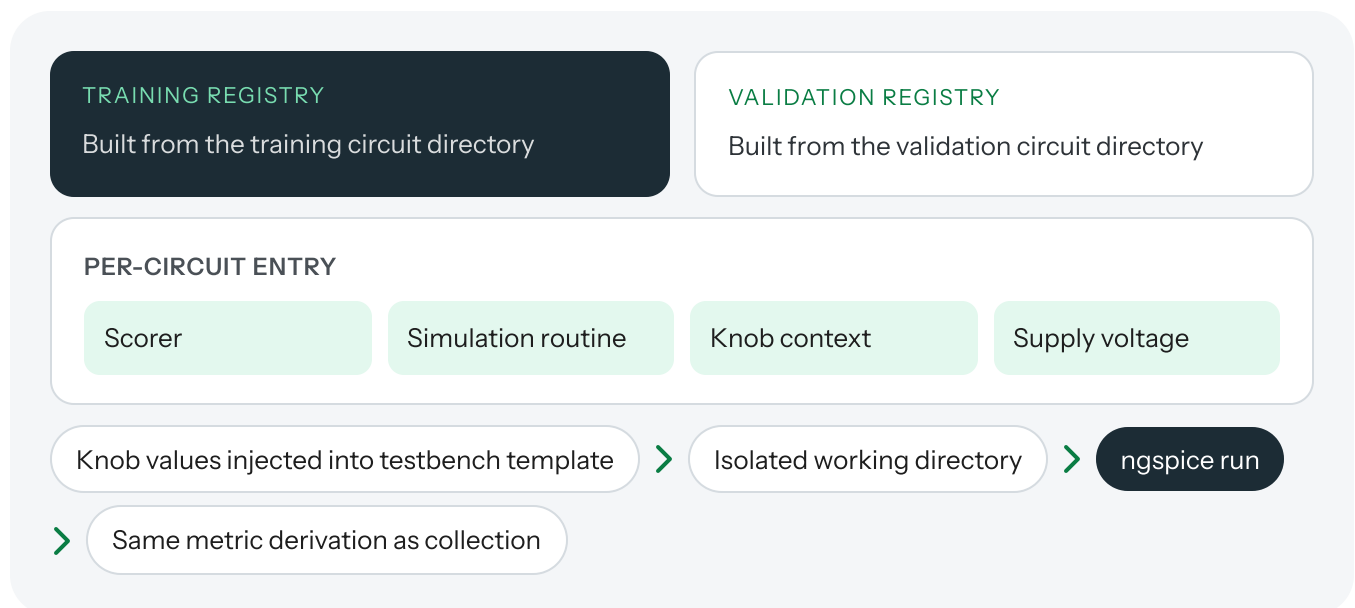


Figure 10. Scoring and simulation resolved per circuit, with training and validation drawn from disjoint circuit sets.

4.5 Training procedure and tuning

GRPO training itself uses a standard single-step group-relative trainer, which builds the training circuit registry described above, initializes each seed's starting state by running that circuit's own simulator and scorer, exposes the per-circuit reward-routing function described above, and then performs the update: sampling multiple completions per seed,

computing the group-relative advantage among them, and applying a policy-gradient update regularized by a Kullback-Leibler penalty back to the frozen supervised-fine-tuning reference policy.

Several tuning choices proved important in practice. The number of completions sampled per seed (the group size) was kept at eight, toward the conservative end of the range reported to be effective elsewhere in the literature, since a larger group size, while generally beneficial for the stability of the advantage estimate, directly multiplies simulation load, which is the binding computational bottleneck in this setting; a very small group size, by contrast, is known to be risky for advantage estimation in general, and is made worse here by the occasional very large penalty values produced by the hard-floor oscillation case. The learning rate was set considerably lower than the range typically reported as effective elsewhere, reflecting a deliberate preference for a slower, more stable early run over a faster but riskier one, with the explicit tuning guidance that a learning rate this conservative should be raised if observed rewards appear flat across training. Sampling temperature was kept at its natural default of 1.0 rather than lowered, since GRPO's advantage signal depends on genuine variance within a sampled group; a supervised-fine-tuned policy sampled at a lower temperature tends to produce near-identical completions within a group, collapsing that variance to zero and producing flat learning regardless of the learning rate chosen, independent of whether the reward itself is discriminating well. When rewards are observed to be flat during training, three distinct causes are checked in order: first, whether sampled actions within a group are themselves identical, which points to temperature being too low; second, whether actions are diverse but the resulting rewards are not, which points to a reward function that fails to discriminate meaningfully between outcomes; and third, whether both actions and rewards are diverse but the policy itself is not visibly changing, which points to either too low a learning rate or too tight a Kullback-Leibler constraint.

DIAGNOSING FLAT REWARDS, IN ORDER

1

Actions within a group are identical

Points to temperature being too low.

2

Actions diverse, rewards not

Points to a reward function that fails to discriminate meaningfully between outcomes.

3**Both diverse, policy not changing**

Points to either too low a learning rate or too tight a Kullback-Leibler constraint.

4.6 A limitation and a path forward: multi-step GRPO

The single-start seeding scheme described in Section 4.2.1, before the intermediate-seed fix of Section 4.2.2 was introduced, exposes a specific and important limitation. Even trained from a single fixed starting state per circuit, GRPO learns a genuinely useful, conditional policy: because the policy reads state features such as topology, current knob positions, and the remaining gap to each target directly from the prompt, a rule of the form "when this metric is meaningfully below target and this knob is near the low end of its legal range, raise it" generalizes to any state sharing those features, not merely to the one state the policy happened to be trained from. Across roughly ninety distinct circuits, the policy is effectively presented with ninety distinct instances of the same underlying "state to best action" learning problem.

The flaw is a distributional one: with only a single fixed starting state per circuit, there is no training signal whatsoever for states more than one step away from that seed. At deployment, however, the optimization loop iterates many times in sequence; only the very first step of a run is drawn from a state the policy was actually trained on, and every subsequent step occurs at a state the policy has never seen during training, with the resulting distribution shift compounding as the run continues, the familiar distribution-shift problem shared by imitation learning and reinforcement learning alike. For a policy intended to run tens of iterations per deployment, single-start seeding systematically under-covers the portion of the state space it will actually be asked to reason about at inference time. The intermediate-seeding scheme described in Section 4.2.2 is adopted specifically to mitigate this: by drawing seeds from multiple points along each teacher's original trajectory, the policy is exposed to genuinely intermediate, partially optimized states using data already collected, without requiring any additional simulation. This mitigation is not complete, since those intermediate states still lie along a single trajectory through state space, a line rather than a genuinely broad cloud of coverage, and a policy that wanders meaningfully off that particular trajectory during its own rollouts may still encounter comparatively under-covered territory. Two more thorough remedies exist beyond the current scope of this work: seeding from randomly perturbed states, which would provide closer to independent, identically distributed coverage of the state space at the cost of substantially more simulation, and multi-step, episodic GRPO, in which an entire multi-step trajectory rather than a single step

is treated as the unit of training, directly optimizing what is ultimately cared about (convergence over many iterations) and inherently correcting the distribution-shift problem, since the policy would then be trained on precisely the intermediate states its own rollouts actually produce. The single-step formulation was nonetheless deliberately chosen to establish the training pipeline first, since it is markedly lower-variance and considerably simpler to implement correctly, credit assignment across a full trajectory is the genuinely hard part of the episodic alternative and demands careful, exactly potential-based reward shaping to avoid introducing new instabilities, and because simulation throughput, already the dominant computational cost across roughly ninety circuits, would multiply directly with trajectory length under an episodic formulation.

REMEDY

Seeding from randomly perturbed states

Closer to i.i.d. coverage, at the cost of substantially more simulation.

REMEDY

Multi-step, episodic GRPO

A whole trajectory as the unit of training, inherently correcting the distribution shift.

05

Hybrid deployment architecture

5.1 Two agents, one task

At deployment, the same underlying task, iteratively resizing a fixed circuit topology to meet its design targets, is served by a hybrid of two agents with deliberately different cost and capability profiles. The first is the domain-specialized model produced by the SFT-then-GRPO pipeline described in Sections 3 and 4: effectively free to run once fine-tuned, but comparatively weaker on the circuits that fall furthest from its training distribution. The second is a proprietary, general-purpose reasoning model, considerably more capable on difficult or unfamiliar circuits, but materially more expensive per call. Rather than committing to one of these two agents for an entire deployment, the design defaults to the inexpensive, fine-tuned model and escalates to the expensive model only for the specific circuits, and specific points within a run, where the inexpensive model demonstrably struggles, turning the choice of which model handles a given circuit into a per-run, evidence-based decision rather than a fixed a priori one.

The pipeline runs in two sequential phases for a given circuit. Phase A, always run first, uses the fine-tuned model for its full allotted step budget, under the exact prompt format it was fine-tuned on. Phase B, run only conditionally, uses the proprietary model, picking up exactly where Phase A left off and consuming whatever step budget Phase A did not use itself, rather than receiving a separately allotted budget of its own. Every step in either phase, regardless of which model is proposing the action, ends the same way: the proposed parameter values are written into the circuit's testbench, the design is re-simulated in batch mode, the resulting measurements are parsed and any derived quantities computed, and the outcome is scored against the design targets, with an invalid or missing measurement surfaced to the model as an explicit repair-mode warning in its next turn rather than silently discarded. The two phases differ only in which model proposes an action and how much surrounding context that model is given, never in how a proposed action is evaluated once made.

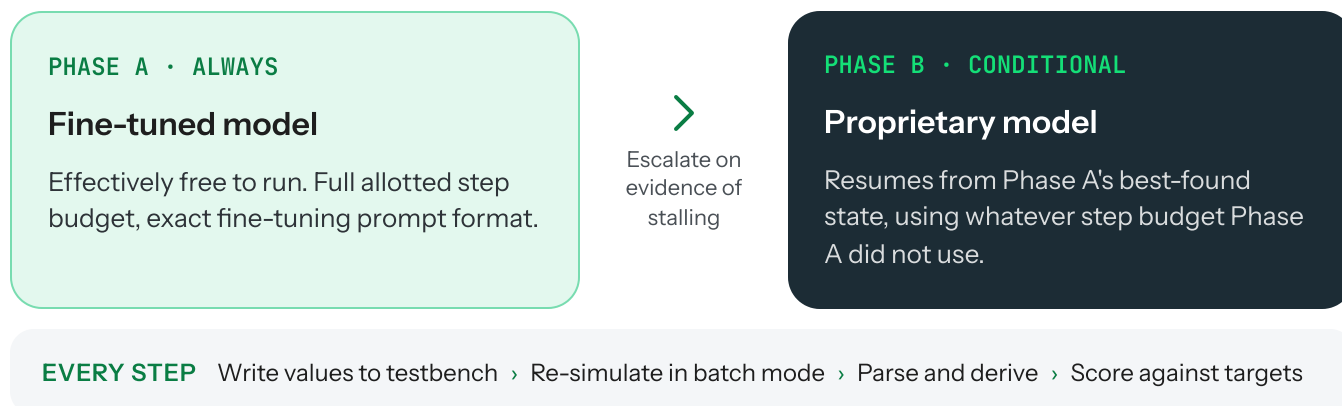


Figure 11. Two agents, one task. The phases differ only in which model proposes an action and how much context it is given, never in how an action is evaluated.

5.2 Escalation triggers

Phase A's loop can end for several distinct reasons, but escalation to Phase B is triggered by only three of them: stagnation, in which the overall error against the design targets has stopped improving meaningfully over a recent window of steps; oscillation, in which the model is visibly cycling between the same few states without making net progress; and repeated parse failure, in which the model's output could not be reliably interpreted as a valid action even after retries. Escalation deliberately does not occur when Phase A converges by fully meeting its design targets, when it exhausts its step budget through ordinary, productive iteration, or when the model itself judges the run complete and elects to hold or stop, since in none of these cases is there any evidence that a stronger, more expensive model would actually improve the outcome. A circuit is only escalated, moreover, if Phase A

still has unused steps remaining in its overall budget; if Phase A consumed its entire allotment before triggering one of the three qualifying conditions above, there is nothing left to hand off, and the run concludes with Phase A's result.

ESCALATES

- Stagnation
- Oscillation
- Repeated parse failure

DOES NOT ESCALATE

- Converged on its design targets
- Budget exhausted through productive iteration
- Model elects to hold or stop

Escalation also requires unused steps remaining in Phase A's overall budget.

5.3 The hand-off from Phase A to Phase B

When escalation does occur, Phase B runs the same underlying loop mechanics as Phase A, but with the proprietary model in place of the fine-tuned one, and with substantially richer surrounding context: the same fixed circuit information presented in full, augmented with an explicit glossary defining what each measured metric means, background reference material on the relevant circuit physics, and explicit behavioral rules governing how the model should prioritize its actions, for instance that circuit stability must be restored before any other target is pursued, that an already-passing target should not be further over-optimized at the expense of others, and under what conditions it is appropriate to revert a change or declare a target structurally unreachable.

Rather than beginning from the circuit's original default parameter values, Phase B's first simulated state is seeded directly from Phase A's own best-found parameters, so that the proprietary model begins its work already at the fine-tuned model's best progress rather than repeating work the cheaper model has already done; parameters that are structurally coupled to another parameter are deliberately left untouched during this seeding step, so that the hand-off does not inadvertently break an intentional coupling between two knobs.

Phase A's entire step history is folded into the proprietary model's first prompt, with three specific adjustments made so that the richer history is not misread. First, every step originating from Phase A is explicitly labeled as belonging to a prior phase, using a numbering scheme distinct from Phase B's own steps, so that the proprietary model does not mistake the fine-tuned model's earlier actions for its own. Second, any self-predictions recorded in Phase A's history are stripped out entirely, since those predictions reflect the fine-tuned model's own calibration rather than the proprietary model's, and leaving them in

place would otherwise produce a spurious "your last prediction was right or wrong" readout misattributed to a model that never made that prediction. Third, the importance score used to decide which historical steps remain visible in a shrinking context window is explicitly zeroed for every carried-over Phase A step, so that a single especially impactful step from Phase A cannot permanently occupy a slot in the visible history; instead, Phase A's steps roll off naturally through the ordinary recency-based windowing mechanism as Phase B accumulates its own history. Every other detail of a carried-over step, what was changed, what happened as a result, and which metrics moved, is preserved unmodified. Going forward from the hand-off point, the proprietary model receives a richer per-step history than Phase A itself was given (an explicit per-step verdict, the specific metrics that moved, and a predicted-versus-actual comparison wherever a prediction was made), earlier trajectory checkpoints, a rolling knob-to-metric sensitivity table, its own previous reasoning reproduced verbatim, and a running account of which of its own hypotheses remain active versus falsified. Any output that cannot be parsed or fails validation is retried, with the specific validation errors appended to the next attempt, up to a configured retry limit, rather than being treated as an immediate hard failure.

01

Labeled as a prior phase

A distinct numbering scheme, so earlier actions are not mistaken for its own.

02

Self-predictions stripped

No readout misattributed to a model that never made the prediction.

03

Importance score zeroed

Phase A steps roll off through ordinary recency-based windowing.

— 06

Benchmark results

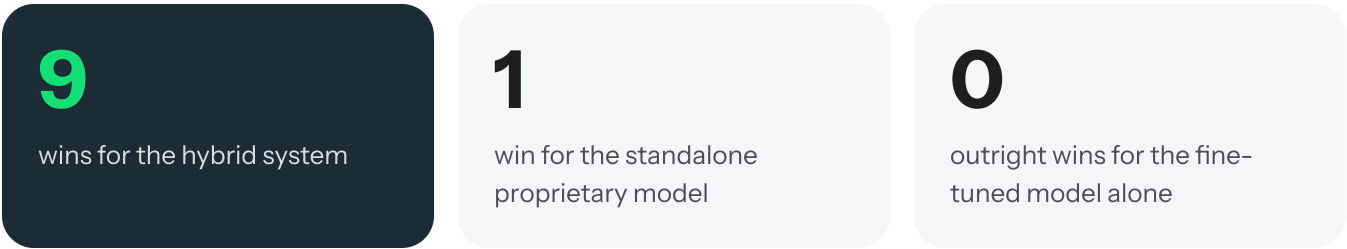
6.1 Evaluation setup

The hybrid architecture was evaluated head-to-head against both single-model baselines, the proprietary model running alone and the fine-tuned model running alone, on a controlled ten-circuit benchmark comprising four LDO voltage regulators and six operational amplifiers, drawn from a broader twenty-circuit corpus. All three configurations were run under identical conditions: the same circuits, the same iteration budget, and the

same sampling temperature. A configuration was judged the winner on a given circuit primarily by the number of design specifications it satisfied, with total proprietary-model token consumption used only as a tiebreaker between otherwise-equivalent outcomes.

6.2 Outcome quality

The hybrid system reached the best available result in the fewest iterations on eight of the ten benchmark circuits, and on three of those eight it was the only one of the three configurations to fully satisfy every design specification, an outcome neither single-model baseline achieved on its own on those same circuits. The proprietary model running alone won outright on exactly one circuit. Overall, the hybrid system matched or exceeded both single-model baselines on nine of the ten circuits, for a head-to-head record of nine wins for the hybrid system, one win for the standalone proprietary model, and zero outright wins for the fine-tuned model running alone.



6.3 Per-circuit breakdown

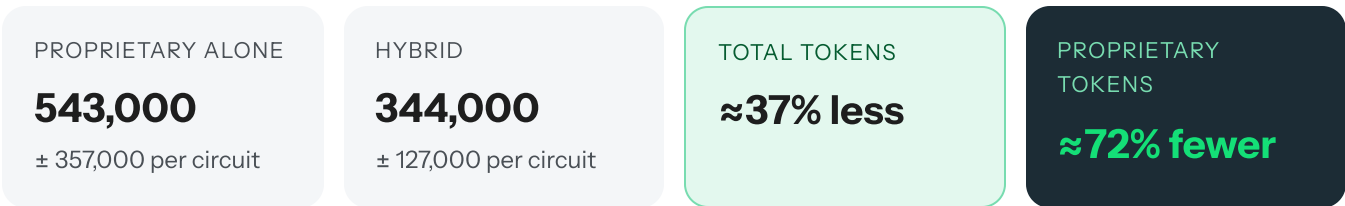
Table 1 summarizes results for each of the ten benchmark circuits. Two rows are particularly worth noting against the design of the escalation trigger itself: circuits 3 and 5 show no escalation at all even though the hybrid system's iteration count matches the fine-tuned model's exactly, meaning the fine-tuned model solved these circuits entirely on its own and the proprietary model was correctly never invoked, so no additional cost was incurred beyond the free run. Circuit 8 is the benchmark's sole loss for the hybrid system: escalation does occur, but the hybrid run stops on stagnation at essentially the same specification count and penalty the proprietary model reached running alone, while consuming more combined tokens (213k from the fine-tuned model plus 298k from the escalated proprietary model) than the proprietary model running alone required (283k); in this case the escalation fired as designed, but the hand-off did not leave the proprietary model enough remaining budget to close the residual gap.

#	Circuit (specs)	Model	Iters	Stop reason	Best specs @ step	Best penalty
1	ag_ldo_papers_2544 (11)	GPT	13	converged	11/11 @13	0.45
		Qwen GRPOv2	13	converged	11/11 @13	0.45
		Hybrid	13	converged	11/11 @13	0.45
2	ag_ldo_papers_2545 (11)	GPT	32	converged	11/11 @32	0.45
		Qwen GRPOv2	9	stagnation	8/11 @3	2.85
		Hybrid	25	converged	11/11 @25	0.45
3	ag_ldo_papers_2546 (11)	GPT	8	converged	11/11 @8	0.40
		Qwen GRPOv2	23	converged	11/11 @23	0.45
		Hybrid	23	converged	11/11 @23	0.45
4	ag_ldo_papers_2549 (11)	GPT	13	stagnation	7/11 @8	2.54
		Qwen GRPOv2	6	oscillation	6/11 @4	5.02
		Hybrid	15	converged	11/11 @15	0.45
5	ag_cmos_opamp_A_985 (17)	GPT	40	budget	13/17 @1	4.27
		Qwen GRPOv2	40	budget	14/17 @2	3.96
		Hybrid	40	budget	14/17 @2	3.96
6	ag_opamp_papers_1788 (17)	GPT	40	budget	14/17 @5	0.94
		Qwen GRPOv2	11	oscillation	13/17 @3	0.82
		Hybrid	30	converged	17/17 @30	0.50
7	ag_opamp_papers_1891 (17)	GPT	15	oscillation	12/17 @1	0.90
		Qwen GRPOv2	17	oscillation	14/17 @10	1.91
		Hybrid	24	stagnation	14/17 @10	1.91
8	ag_opamp_papers_1893 (17)	GPT	12	oscillation	16/17 @6	0.56
		Qwen GRPOv2	22	oscillation	16/17 @19	0.57
		Hybrid	39	stagnation	16/17 @28	0.56
9	ag_opamp_papers_1914 (17)	GPT	6	oscillation	13/17 @1	3.17
		Qwen GRPOv2	11	oscillation	14/17 @10	1.26
		Hybrid	18	stagnation	14/17 @10	1.27
10	ag_opamp_papers_1918 (17)	GPT	10	topology	16/17 @10	0.50
		Qwen GRPOv2	22	converged	17/17 @22	0.55
		Hybrid	22	converged	17/17 @22	0.55

Table 1. Per-circuit benchmark results (best specs satisfied / total specs, and best penalty reached).

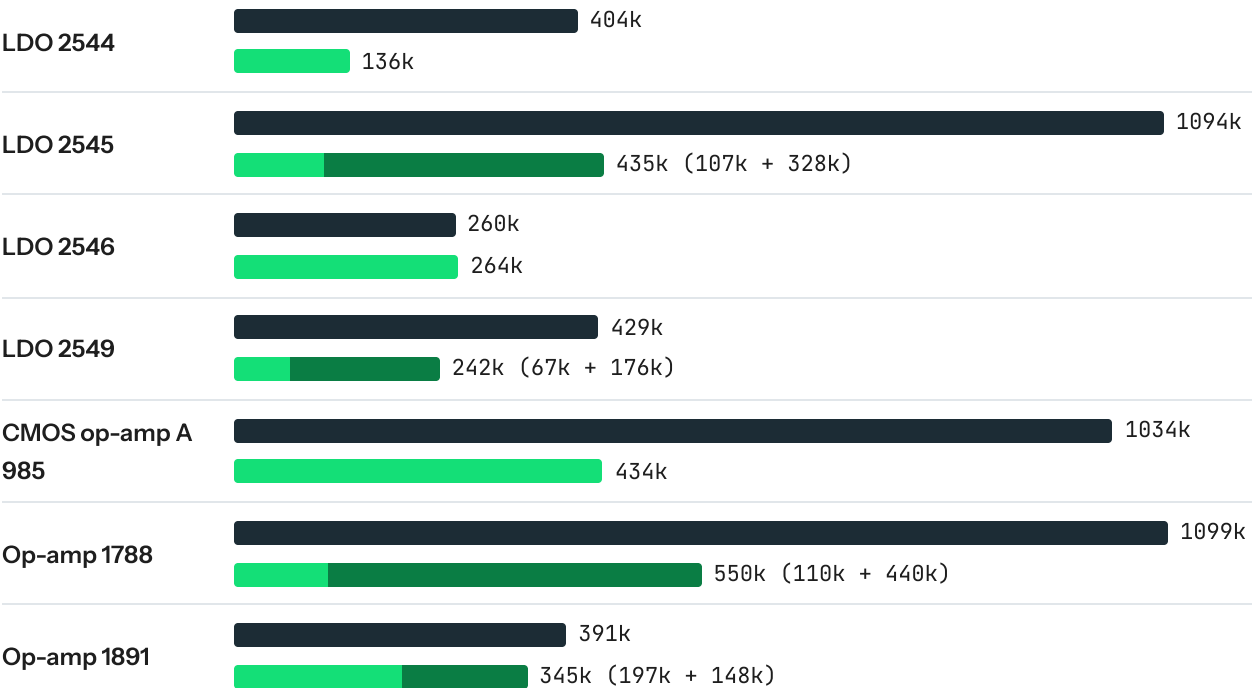
6.4 Token economics

Averaged across all ten circuits, a pure proprietary-model run consumed $543,000 \pm 357,000$ total tokens per circuit, compared with $344,000 \pm 127,000$ total tokens for the hybrid approach, a reduction of roughly 37% in total token usage. Because the large majority of tokens consumed by a hybrid run come from the free-to-run fine-tuned model rather than the proprietary one, the reduction specifically in proprietary-model-billed tokens is considerably larger, approximately 72% fewer proprietary-model tokens than running the proprietary model alone. This follows directly from the escalation design described in Section 5: the proprietary model is invoked only on the subset of circuits where the fine-tuned model's own trajectory stalls, and even in those cases it resumes from the fine-tuned model's best-found point rather than re-optimizing the circuit from scratch. Figure 12 shows per-circuit token consumption broken down by model and phase.



Token consumption per circuit: GPT-5.5 vs hybrid

■ GPT-5.5 ■ Hybrid: Qwen-GRPO-v2 ■ Hybrid: GPT-5.5



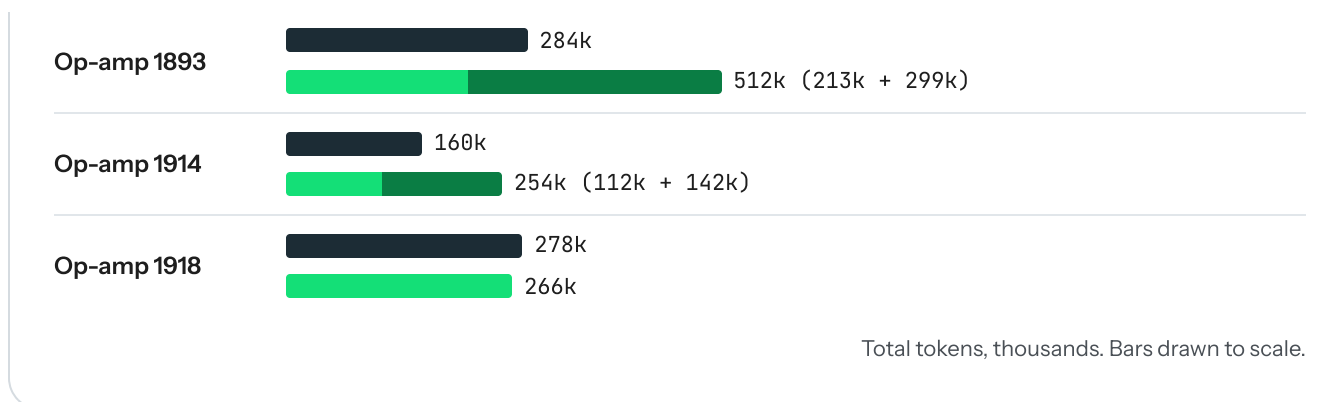


Figure 12. Per-circuit total token consumption: the proprietary model alone (dark) versus the hybrid run's fine-tuned-model component (bright green) plus its proprietary-model escalation component (dark green), where escalation occurred. Bars with no dark-green segment (LDO 2544, LDO 2546, CMOS op-amp 985, op-amp 1918) are cases where the fine-tuned model converged on its own and the proprietary model was never invoked.

6.5 Net takeaway

The hybrid system achieves this cost reduction without a corresponding quality trade-off in aggregate: it is strictly better than both single-model baselines, achieving full specification satisfaction where neither single-model configuration did so alone, on three of the ten circuits, and it is never meaningfully worse than the best single-model result on any other circuit except the one case noted in Section 6.3. On that single circuit where the proprietary model alone won outright, the hybrid system reached the same stagnation failure mode as the standalone fine-tuned model before escalating, and did not fully recover within its remaining step budget after the hand-off. This is treated here as a tuning question for the escalation policy's thresholds and remaining-budget allocation, rather than as evidence against the hybrid approach in general, and is noted explicitly as an open limitation for future refinement.

07

Conclusion

This paper described how structured reasoning traces of analog circuit sizing are converted into a deployable optimization system through a two-stage training pipeline and a hybrid deployment architecture. A shared prompt-rendering component keeps every stage, trace collection, supervised fine-tuning, GRPO rollout, and deployment, on a single consistent input distribution. Supervised fine-tuning installs a correct, minimal output schema and a coherent reasoning style from teacher demonstrations, with particular attention paid to a

subtle data-quality issue in how hypothesis references were rendered. GRPO then trains the resulting policy beyond its teacher using repeated circuit simulation as a ground-truth reward, with a specific and consequential correction applied to prevent the policy from learning to exploit silently-missing measurements rather than genuinely meeting design specifications. At deployment, this fine-tuned model is paired with a considerably more expensive, general-purpose proprietary model in a two-phase hybrid architecture that escalates only on clear, legible evidence of stalling and resumes from the fine-tuned model's best-found state rather than restarting. On a ten-circuit benchmark, this hybrid system matched or exceeded both single-model baselines on nine of ten circuits, achieved full specification satisfaction on three circuits that neither single-model baseline reached alone, and reduced proprietary-model token consumption by approximately 72% relative to running the proprietary model by itself. Together, these results suggest that a domain-specialized small model, trained with a two-stage imitation-then-reinforcement-learning pipeline on carefully hardened reward signals, combined with an evidence-based rather than fixed escalation policy to a stronger general-purpose model, is an effective and considerably more economical strategy for deploying capable analog circuit optimization at scale, while leaving a clear, well-characterized path, principally multi-step episodic reinforcement learning and a more tuned escalation policy, for further improvement.

THE RESULT

A domain-specialized, RL-trained small model, combined with a disciplined and evidence-based escalation policy rather than a fixed model choice, can recover most of the quality of a much larger general-purpose model at a fraction of its cost.