

— ANALOG DESIGN AGENTS

Trace Collection for LLM-Driven Analog Circuit Optimization

Data, pipeline, and a two-round collection protocol on the **SKY130 / ngspice stack**

AUTHOR

Neurologic AI

PUBLISHED

July 2026

STACK

SKY130 / ngspice

ABSTRACT

Training or evaluating a language model as an analog design agent requires something that does not exist publicly: a corpus of complete, simulator-grounded optimization episodes in which a model repeatedly proposes transistor sizings, sees what the simulator says, and revises its beliefs. This paper documents the construction of such a corpus for Project Micron. We assembled 103 transistor-level CMOS circuits (20 multi-stage amplifiers and LDOs from AnalogGym and 83 circuits converted from AnalogGenie) into a uniform three-file representation,

retargeted every one onto the SkyWater SKY130 1.8 V PDK, and drove each through a closed propose-simulate-score loop against ngspice. Because datasheet-class specifications are frequently unreachable for a given topology, collection ran in two rounds. Round 1 optimized against the original specification and, on every non-converged exit, applied a bounded deterministic relaxation of the failing targets before re-running; those phase traces were then mined to derive a per-circuit achievable target set. Round 2 re-optimized against the finalized targets with relaxation disabled and a larger step budget. The campaign produced 304 trace files and 4,880 step records, each carrying the full before/after parameter vector, the raw and derived measurement dictionary, a per-target penalty decomposition, the model's numeric predictions, its falsifiable hypothesis state, and a native reasoning summary. We describe the data and its file formats, the PDK and simulator integration, the two-round protocol and its exact relaxation arithmetic, and the measured outcomes of the campaign.

103

Transistor-level
CMOS circuits

304

Trace files

4,880

Step records

34%

Of round-2 runs
converged

— 01

Introduction

An analog circuit does not have a loss function. It has a netlist, a process technology, a testbench, and a specification sheet, and the only way to learn whether a proposed sizing is good is to run a simulator. This makes analog sizing an attractive target for language-model agents: the search is discrete, the feedback is cheap relative to silicon, and the domain knowledge that guides good moves is exactly the kind of textual, causal knowledge that language models encode. It also makes the problem hard to study, because the artifact one needs in order to train or evaluate such an agent is not a static dataset of input-output pairs. It is a trajectory: a sequence in which a proposal is made, a simulator adjudicates it, and the next proposal is conditioned on that adjudication.

No public source ships the four things required to generate such trajectories at scale. A netlist a model can read, per-knob legal ranges, a testbench that measures the metrics of interest, and a target specification: benchmark suites supply at most the first and third. AnalogGym provides netlists and measurement recipes but no knob ranges and no targets. AnalogGenie provides a very large corpus of topologies with neither. Knob ranges must therefore be derived from the process design kit, and targets must be synthesized against a real datasheet class.

This paper documents what we built to close that gap. The deliverable is a corpus of optimization traces over 103 circuits, grounded in the SkyWater SKY130 130 nm PDK and adjudicated by ngspice. Its distinguishing properties are as follows.

Simulator-grounded, not synthetic.

Every measurement in every row is the output of an actual `ngspice` batch run on a rendered SKY130 netlist. No metric is imputed.

Complete episodes, not isolated moves.

A trace records the whole run: the baseline operating point, every step, the score trajectory, and the reason the run terminated.

Auditable scoring.

Every step carries a per-target decomposition (measured value, bound, pass/fail, raw penalty, priority weight, weighted penalty), so a consumer can recompute the scalar objective or replace it entirely.

Recorded belief state.

Each step stores the model's numeric predictions, its active and falsified hypotheses, and a per-parameter mechanism claim, making the trace a record of *reasoning quality*, not just outcome.

Feasible targets, derived not guessed.

The two-round protocol replaces unreachable datasheet specifications with targets the circuit demonstrably achieved, so a converged run is a meaningful event.

The circuit corpus

2.1 Provenance and scope

The corpus of interest comprises 103 circuits drawn from two upstream sources. Table 1 gives the counts.

Set	Circuits	Knobs (min / med / max)			Targets (med)
AnalogGym	20	21	29	66	17
AnalogGenie	83	13	61	113	11
Total	103	13	n/a	113	n/a

Table 1. The two source sets. Knob and target counts are per circuit.

The **AnalogGym** set is the hand-curated core: sixteen multi-stage operational amplifiers from the frequency-compensation literature, each in its `_Pin_3` variant (`Alfio_RAFFC` , `Fan_SMC` , `HoiLee_AFFC` , `Leung_DFCFC1/2` , `Leung_NMCF` , `Leung_NMCNR` , `Peng_ACBC` , `Peng_IAC` , `Peng_TCFC` , `Qu2017_AZC` , `Ramos_PFC` , `Sau_CFCC` , `Song_DACFC` , `Tan_CLIA` , `Yan_AZ`), plus four low-dropout regulators. These carry the richest specification blocks (a median of 17 simultaneous targets) and the most carefully authored testbenches.

The **AnalogGenie** set is 83 circuits sampled from a converted pool of 3,062. Their knob counts run substantially higher than AnalogGym's (a median of 61 and up to 113 independently tunable parameters) because the conversion exposes every device dimension of every transistor rather than grouping matched devices behind shared symbols.

2.2 Circuit type distribution

Table 2 classifies the 103 circuits by name-derived function. The mix is deliberately unbalanced toward regulators and comparators, because those two families exercise measurement paths (DC sweeps, load transients, threshold crossings) that pure amplifier circuits never touch, and therefore stress the metric-extraction layer harder.

Family	n	Family	n
Low-dropout regulator (LDO)	35	Fully differential op-amp	2

Family	n	Family	n
Comparator	25	High-performance op-amp	2
General / multi-stage amplifier	17	Output stage	2
Op-amp (<code>opamp_papers</code> , <code>opamp_n</code>)	10	Differential amplifier	2
CMOS op-amp	3	Other (DAC, freq. response, stability, design study)	4
CMOS amplifier	1		

Table 2. Circuit-type distribution across the 103 circuits.

03

On-disk representation

3.1 The three-file circuit format

Every circuit in the corpus, regardless of provenance, is normalized into a single directory containing three authored files and one derived cache directory:

File	Role
<code>topology.sp</code>	PDK-neutral subcircuit definition. Device instance prefixes and model names are placeholders; all sizings are <code>{knob}</code> expressions.
<code>testbench.sp</code>	Stimulus, analyses, and <code>.meas</code> statements. Holds the live <code>.param</code> block that the optimizer rewrites each step.
<code>circuit.json</code>	Machine-readable contract: knob list, knob families, fixed parameters, goal targets, priority weights, canonical defaults, baseline measurement, strategy tags, physics notes.
<code>circuit_final.json</code>	Derived after round 1: the achievable goal block plus its provenance. Auto-preferred over <code>circuit.json</code> 's goal when present.

File	Role
<code>_pdk_sim_cache/</code>	Derived, git-ignored. Holds the PDK-rendered copy that <code>ngspice</code> actually executes.

Table 3. Files in a circuit directory.

The separation matters for reproducibility: `topology.sp` never mentions SKY130. A device line reads

```
{ {INST:PFET_STD} }m11 VOUT net050 VDDA VDDA { {MODEL:PFET_STD} }
L={MOSFET_11_1_L_gmf_P MOS} W={MOSFET_11_1_W_gmf_P MOS*1}
M={MOSFET_11_1_M_gmf_P MOS}
```

so a topology can be retargeted by swapping the PDK manifest, and knob names carry their own semantics: device index, dimension (`L` / `W` / `M`), functional role (`gmf` = feedforward transconductor), and channel type. Devices intended to match are expressed as multiples of a shared knob inside the topology, so matching is structural rather than a convention the optimizer must maintain.

3.2 The `circuit.json` contract

`circuit.json` is the interface between the corpus and the pipeline. Its keys are as follows.

Key	Meaning
<code>name</code> , <code>topology_tag</code>	Identifier and topology class (e.g. <code>recycling_amplifier_with_feedforward_compensation</code>). <code>description</code> adds prose on the analyses run.
<code>supply_v</code> , <code>load_pf</code>	Nominal supply and load capacitance.
<code>knobs</code>	Ordered list of tunable <code>.param</code> names.
<code>knob_families</code>	Per knob: PDK device family (<code>NFET_STD</code> , <code>PFET_STD</code> , ...) and kind (<code>W</code> , <code>L</code> , <code>M</code> , <code>current</code> , <code>scalar</code>). Determines which legality rules apply. <code>matching_groups</code> lists knob sets that must move together.
<code>fixed_params</code>	Parameters the optimizer must not touch (<code>VDD</code> , <code>VSS</code> , <code>C_load</code> , <code>R_load</code> , <code>VCM_ratio</code> , step amplitudes).

Key	Meaning
<code>goal</code>	<code>{metric: {min max: value} }</code> . Two-sided targets carry both.
<code>goal_weights</code>	Per-metric priority weight, default 1.0. Scales the progress signal only; never affects pass/fail.
<code>datasheet_specs</code>	The reference part and the raw datasheet numbers the goal was synthesized from.
<code>knob_examples</code> , <code>baseline</code>	Canonical defaults, and the measured dictionary plus provenance prose at those defaults.
<code>strategy_tags</code>	Named design moves available for this topology (<code>bias_recovery</code> , <code>compensation_cap_tune</code> , <code>psrr_decoupling</code> , ...).
<code>physics_notes</code>	Markdown: reference-part mapping, testbench architecture, governing equations, lever guidance. Injected into the model's context.

Table 4. `circuit.json` schema.

Targets are synthesized against a real part rather than invented. For the amplifier circuits the reference is the Texas Instruments TLV9001 (SBOS703) at 1.8 V, $R_L = 10\text{ k}\Omega$ to mid-supply, $C_L = 5\text{ pF}$. The resulting goal block for `Alfio_RAFFC_Pin_3` contains 17 targets: `gain_db` ≥ 100 , `ugbw_hz` $\geq 1\text{ MHz}$, `i_supply_a` $\leq 77\text{ }\mu\text{A}$, `power_mw` ≤ 0.139 , `phase_margin_deg` ≥ 78 , `vos_uv` $\in [-1600, 1600]$, `ib_pos_pa` and `ib_neg_pa` $\in [-5, 5]$, `cmrr_db_100` and `cmrr_db_1k` ≥ 68 , `psrr_db_100` and `psrr_db_1k` ≥ 80 , `slew_pos_v_per_us` and `slew_neg_v_per_us` ≥ 2 , `settle_time_us` ≤ 3 , and `vout_swing_high_mv` and `vout_swing_low_mv` ≤ 20 .

Two features of this block deserve emphasis. First, the recorded baseline is usually *broken*, not merely suboptimal: at canonical defaults this circuit measures `gain_db` = -19.15 with `ugbw_hz` and `phase_margin_deg` both unmeasurable, because the topology does not bias under TLV9001 conditions. The first job of any run is therefore bias recovery, not spec tuning, and the traces contain that recovery. Second, a median of 17 simultaneous constraints spanning gain, bandwidth, stability, power, offset, rejection, slew, settling, and swing means the objective is genuinely multi-dimensional and the trade-offs are real.

Conversion: from raw sources to the three-file format

Neither upstream source ships anything the pipeline can consume directly. This section documents the two converters that produce the format of Section 3; the assumptions they encode determine what the traces can and cannot say.

4.1 What the raw sources provide

AnalogGym is organized by block type: Amplifier, Low Dropout Regulator, Phase-Locked Loop, Charge Pump, Sensing Front End, Voltage Reference. The amplifier family, which is what we use, ships three artifacts per circuit: a `spice_netlist/<name>` file holding a `.subckt` already written against sky130 model names with sizes as quoted parameter references (`l='MOSFET_11_1_L_gmf_PMO5'`), a `design_variables/<name>` file holding a `.PARAM` continuation block of canonical values, and two shared testbenches (`TB_Amplifier_ACDC.cir` , `TB_Amplifier_Tran.cir`) in which the circuit under test is hard-coded by an `.include` line and five instance lines. LDOs follow the same pattern with a different parameter convention (`W_M1` rather than `MOSFET_<id>_W_<role>_<type>`).

AnalogGenie is one directory per circuit under `Dataset/<N>/` holding `<N>.cir` , `Port<N>.txt` , `Pagenum<N>.txt` (the source citation), and page images. 3,351 such directories exist. A device line reads

```
M5 (net032 VIN2 IB1 VSS) nmos4
```

Connectivity and a device class, and nothing else. Table 5 sets out what each source supplies against what the pipeline needs.

Two absences dominate the design: neither source supplies knob ranges, so those are resolved from the PDK manifest at optimize time rather than baked into the corpus, and neither supplies targets, so both converters key their goal blocks off a real datasheet part rather than a per-circuit guess.

Requirement	AnalogGym	AnalogGenie	Where it comes from instead
Machine-readable netlist	yes	yes	n/a

Requirement	AnalogGym	AnalogGenie	Where it comes from instead
Device sizes / bias values	yes	no	converter defaults (deliberately poor)
PDK model binding	sky130	no	MODEL_TO_FAMILY / family map
Per-knob legal ranges	no	no	the PDK manifest, at optimize time
Batch measurement harness	partial	no	generated per family
Target specification	no	no	TI datasheet tables

Table 5. What the raw sources ship, against the four things a trace-generating run requires.

4.2 AnalogGym: retargeting an already-sized netlist

Because AnalogGym netlists already carry sizes and sky130 model names, the work is normalization rather than synthesis.

Topology. Each `xm ...` instance line is matched by regex and rewritten: the model name is looked up in a seven-entry table mapping sky130 subcircuit names to abstract families (`sky130_fd_pr__nfet_01v8` $\rightarrow$ `NFET_STD` , and so on), and the instance prefix and model become `{ {INST:family} }` and `{ {MODEL:family} }` . Quoted size attributes are unquoted into brace form, `l='X'` $\rightarrow$ `L={X}` , so ngspice resolves them against the live `.param` block. Multiplier expressions are preserved verbatim, which is what keeps device matching structural.

Passive substitution. The sky130 `cap_mim` and `res_high_po` model files declare statistics with Spectre-syntax `vary ... dist=gauss` blocks ngspice cannot expand, so calls to them are replaced with lumped primitives (roughly 200 fF per MIM unit cell and 2.7 k Ω per high-poly resistor unit), each scaled by the original multiplicity attribute, so the optimizer keeps the knob even though the device model is gone.

Parameters and units. The `design_variables` block is read with SPICE `+` continuations collapsed. AnalogGym relies on `.option scale=1u` , so its `W` / `L` values are bare micrometre floats; we do not use `scale` , so each is rescaled to an explicit SI-suffixed form (`1` $\rightarrow$ `1u` , `0.5` $\rightarrow$ `500n`). Multiplicities are coerced to integers; capacitances and currents already

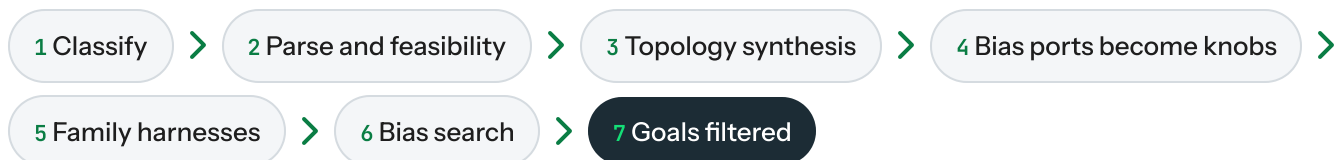
carry suffixes and pass through. Four AnalogGym globals (`CLOAD` , `VCM` , `MC_MM_SWITCH` , `MC_PR_SWITCH`) are dropped from the knob list because the generated testbench owns them.

Knob classification. Each surviving parameter is assigned a `{family, kind}` pair from its name (`_NMOS` / `_PMOS` selects the device family, an `_M_` infix or `M_` prefix marks a multiplicity, `_W_` and `_L_` mark geometry, a `CURRENT_` prefix or `_BIAS` suffix a bias current, `CAPACITOR_` a capacitance), and that pair later selects which legality rule applies.

Testbench and targets. The two interactive AnalogGym harnesses (which plot rather than report, one circuit at a time) are replaced by a single batch harness per circuit, using a 1 TH inductor and 1 TF capacitor to close the DC feedback loop while leaving the AC path open, so open-loop gain is measurable without hand-biasing. A second pass then promotes each circuit to the seventeen-metric multi-analysis rig of Section 5, resets knobs to canonical defaults, recaptures the baseline, and rewrites the goal block from a datasheet: TLV9001 for fifteen circuits, TLV9061 for `Leung_NMCF` , whose 100 pF load needs the faster part. Every goal value cites the datasheet section it came from; none is padded.

4.3 AnalogGenie: synthesizing everything

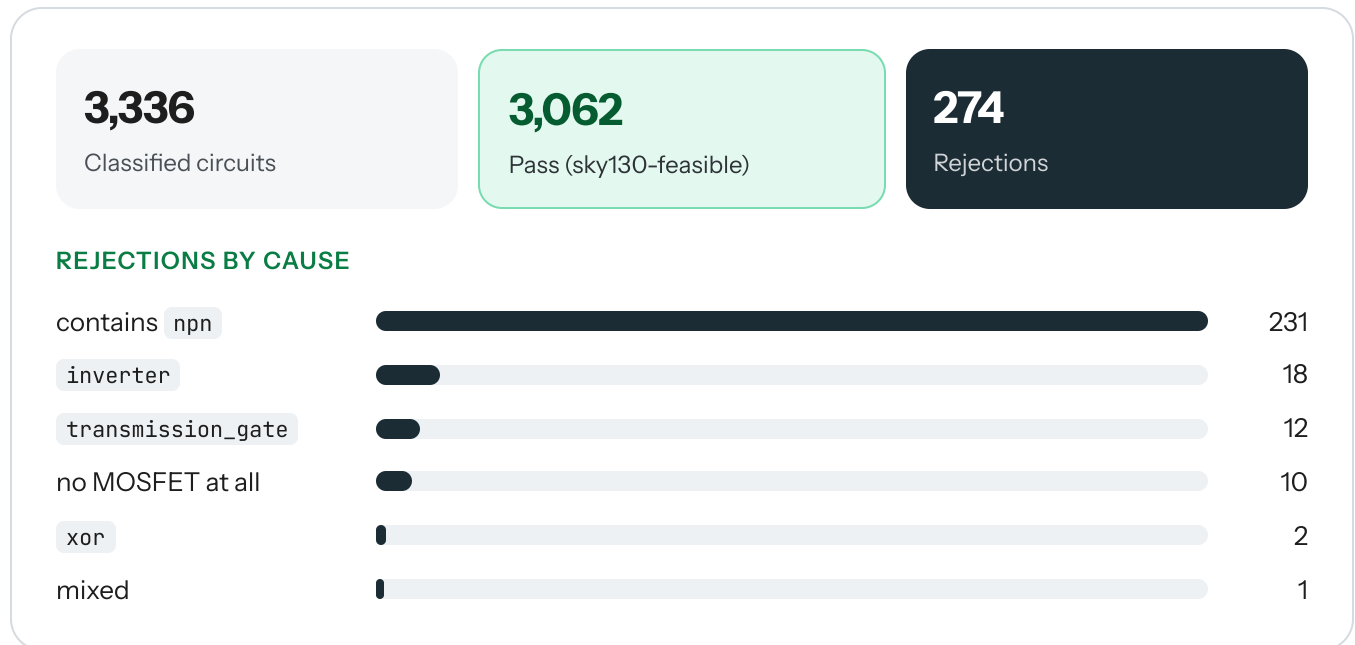
AnalogGenie gives topology and nothing else, so the converter must invent sizes, a bias point, a PDK binding, a harness, and a spec, in that order, and each step is filtered.



Step 1: classification by provenance. The dataset tags each circuit only by the page range of the textbook or paper it was drawn from. Those ranges are transcribed into a table of `(label, lo, hi, family, in_scope, ti_target)` entries, so a circuit's functional class inherits the provenance of its source text rather than a guess about its netlist: Razavi 2e pages 281 to 336 become `opamp` , 160 to 183 `amp_freq_response` , 461 to 492 `lna` (RF). Thirteen families are recognized; six map to a datasheet anchor (Table 6).

Step 2: parse and feasibility. A parser reads `<N>.cir` into typed device records and `Port<N>.txt` into a port list, normalizing sixteen raw device tokens (`nmos4` , `pmos3` , `resistor` , `npn` , `transmission_gate` , ...) into nine kinds. A circuit is sky130-feasible only if it contains at least one MOSFET and no device the process cannot realize. Of 3,336 classified circuits, 3,062 pass; the 274 rejections break down as 231 containing `npn` (sky130 ships only

pnp), 18 inverter , 12 transmission_gate , 10 with no MOSFET at all, 2 xor , and 1 mixed. The scan is written to _index.json with class, family, feasibility, device counts, kinds, ports and port-role tallies; a companion _curated.json ranks candidates by a testbench-friendliness heuristic rewarding a differential input pair, a single primary output, present supply and ground rails and a moderate device count.



Step 3: topology synthesis with knobs. Every MOSFET receives three fresh knobs, `<inst>_W/_L/_M` ; three-terminal devices get a bulk connection defaulted by channel type. Resistors, capacitors and inductors become primitives with a single value knob each; diodes get a local `.model` ; pnp maps to the sky130 5.5 V PNP. Because paper-derived netlists reuse reference designators (`MM0` can appear three times), instance names are uniquified so each device gets distinct knobs, which is why AnalogGenie knob counts reach 113 where AnalogGym's median is 29.

Step 4: bias ports become knobs. Unlike AnalogGym's self-biased subcircuits, AnalogGenie exposes bias rails and reference currents as external ports. The testbench drives each from a `.param` , and those become knobs: `BIAS_VB1` as a scalar bounded to $[0, 1.8]$ V, `BIAS_IB1` as a current, plus `VIN_DC` for a single-input block. The optimizer therefore tunes device sizes *and* the operating point: a materially harder search, and the reason these traces contain so much bias-recovery behaviour.

Wiring those ports requires inferring what they are. A name-pattern classifier assigns each port a role (ground, supply, input, output, bias current, bias voltage, reference, clock), tying anything unclassifiable to mid-rail so the block still biases. Bias-current direction is then

inferred structurally, from whether the port sits on an NMOS or PMOS source or drain, and bias-voltage polarity from whether the port gates more PMOS or NMOS devices; a high-side and a low-side rail must be driven in opposite directions.

Step 5: family-specific harnesses. One testbench for all circuits would measure the wrong things, so six are generated (Table 6). The LDO case is worth singling out: the converter locates the resistor-divider feedback tap that also drives an error-amplifier gate, splits that node into `FBDIV` and `FBAMP` ports, and has the testbench bridge them with a 0 V series source (DC-transparent, so regulation is unaffected, but breaking the loop for an AC return-ratio sweep). That is what makes loop gain, unity-gain bandwidth and phase margin measurable on a closed-loop regulator at all.

Step 6: bias search and the bad-baseline filter. Canonical defaults are deliberately poor but biasable: $W = 2\ \mu\text{m}$, $L = 0.5\ \mu\text{m}$, $M = 2$, $R = 10\ \text{k}\Omega$, $C = 1\ \text{pF}$, $V_B = 0.9\ \text{V}$, $I_B = 10\ \mu\text{A}$. The converter sweeps a small grid: input polarity $\times$ two polarity-aware bias levels (0.8/0.6 V for PMOS-gated rails, 1.0/1.2 V for NMOS-gated) $\times I_B = 20\ \mu\text{A}$, simulating each and ranking by a family-specific aliveness scalar (DC gain for amplifiers, output swing for comparators and switched-capacitor blocks, regulation error for references and LDOs). A circuit is retained only if it is alive *and* still fails at least one target: dead or railed circuits are discarded, and so are circuits whose baseline already passes everything, because neither yields a climbable trajectory. Every decision is logged to `_convert_report.json` with its verdict string and the measured dictionary at defaults.

Step 7: goals filtered to what the rig can measure. The family's datasheet goal block is finally intersected with the metric names the generated testbench actually emits, scanned from its `.meas` and `echo` statements. A target the harness cannot produce is dropped rather than left to score as a permanent miss, and a circuit for which nothing measurable survives is recorded `no_measurable_goal` and not shipped. Hence the per-circuit target count varies across the set (17 for a fully instrumented op-amp, 11 for an LDO, 5 for a comparator), and the median is 11 against AnalogGym's 17.

Family	Anchor	Harness	Targets
amplifier, opamp	TLV9001	<code>.op</code> + 3 AC sweeps + <code>.dc</code> + <code>.tran</code>	17
ldo	TPS7A02	closed-loop regulation, line/load sweeps, PSRR, load transient, broken-loop AC	11

Family	Anchor	Harness	Targets
comparator	TLV3201	DC sweep (swing, offset, hysteresis) + transient (propagation delay)	5
reference	REF_GEN	DC supply sweep (line regulation) + AC supply injection (PSRR)	5
switched_cap	SC_SAMPLER	non-overlapping clocks + transient (swing, settling)	5
rf, power, oscillator, filter, pll, data_conv, misc	GENERIC	powers-up / responds check	3

Table 6. AnalogGenie family → generated harness and datasheet anchor.

4.4 Synthesizing `circuit.json`

The contract of Section 3 is assembled last, because several of its fields are functions of the topology and testbench the earlier steps produced. Table 7 records where each field comes from in each converter.

Field	AnalogGym	AnalogGenie
<code>name</code>	source circuit name	<code>ag_<class>_<id></code>
<code>topology_tag</code>	hand-curated table, 17 entries, circuit → compensation scheme	the page-range class label
<code>knobs</code>	surviving <code>design_variables</code> names, in file order	emitted in device order during topology synthesis
<code>knob_families</code>	name-pattern classifier	assigned at synthesis; bias-voltage knobs also carry rail bounds
<code>knob_examples</code>	rescaled canonical values	the poor defaults chosen by the bias search
<code>fixed_params</code>	fixed list owned by the harness	parsed back out of the generated testbench
<code>goal</code>	datasheet table for the assigned part	datasheet table <code>n</code> what the harness emits

Field	AnalogGym	AnalogGenie
<code>goal_weights</code>	n/a	datasheet weights, filtered to surviving keys
<code>datasheet_specs</code>	raw part numbers	raw part numbers
<code>baseline</code>	imported, then replaced by a fresh simulated capture	captured by running ngspice at defaults
<code>strategy_tags</code>	amplifier vocabulary	per-family vocabulary
<code>physics_notes</code>	templated scaling laws + reference part	templated + explicit conversion provenance

Table 7. Provenance of every `circuit.json` field.

Three of these rows are doing real work rather than bookkeeping.

`fixed_params` is derived, not declared.

On the AnalogGenie side the converter re-parses its own generated testbench for every `.param` and `.csparam` name and subtracts the knob list, using the same parser the structural validator uses. The consequence is that the two can never disagree: any quantity the harness declares but the optimizer must not touch (supply, common-mode ratio, load, step amplitudes, clock periods) is listed as fixed automatically, so a harness change cannot silently open a parameter for editing.

`goal` is a function of the harness, through the runtime's own dependency map.

A datasheet target survives only if every upstream raw `.meas` key it depends on appears in the generated testbench, and the dependency map consulted is the same `DERIVED_METRICS` table the runtime uses to compute the metric (Section 5). Goal and measurement therefore cannot drift apart, and no target can become a permanent scoring floor because the rig was never able to produce it. `goal_weights` is filtered to the surviving keys, and a circuit for which nothing survives is not written at all.

baseline is measured, not asserted.

The AnalogGenie converter runs ngspice on the freshly written testbench at the canonical defaults and stores what comes back. Because `meas` and `echo` print as each analysis completes, results from an earlier block survive even when a later one diverges and returns non-zero; the capture is only treated as a failure when nothing measurable came out at all. The `baseline.source` string then records the reference part, its document number, and a bias-state sentence chosen by whether any goal metric was measurable: either "Most ... targets are unmet; the optimizer's job is to close the gaps" or "At canonical defaults the topology does not produce usable measurements: `bias_recovery` is the first task". On the AnalogGym side the first pass imports the upstream project's best tuned metrics as informational only, translating names into ours (`dcgain` → `gain_db` , `gbp` → `ugbw_hz` , `phase_in_deg` → `phase_margin_deg` , `power` → `power_mw` in mW); the datasheet pass then discards it and recaptures the baseline under our own harness, which is why the shipped baselines are frequently worse than the upstream figures: they are honest measurements of the canonical starting point rather than of a tuned one.

4.5 Why this matters for the traces

Three properties of the corpus follow directly from the conversion. Baselines are weak by construction rather than by accident, so every trace begins with real work to do. Targets are datasheet-anchored and measurability-filtered, so a failing target reflects the circuit rather than the harness. And knob semantics are uniform (every knob carries a family and kind), so one validation gate and one PDK bounds check serve 103 circuits from two sources sharing no naming convention.

— 05

PDK grounding and simulator integration

5.1 The PDK manifest

The active process is described by a manifest, `pdk/sky130/pdk.json` , which is load-bearing in three distinct ways: it supplies the model names and instance syntax used to render a netlist, the legality envelopes used to validate a proposed sizing, and the supply voltage used to derive power.

01

Model names and instance syntax used to render a netlist

02

Legality envelopes used to validate a proposed sizing

03

Supply voltage used to derive power

Seven device families are declared. Five are real 1.8 V MOSFET families: `NFET_STD` (`→ sky130_fd_pr__nfet_01v8`), `PFET_STD`, `NFET_LVT`, `PFET_LVT`, `PFET_HVT`, each specifying its SPICE subcircuit name, an instance prefix of `x` (SKY130 devices are subcircuit wrappers around BSIM4 models, not primitives), pin order, default bulk connection, and `nf` as the width multiplier. Two further families, `CAP_MIM` and `RES_HIGH_P0`, are metadata-only labels: the LDO testbenches instantiate those elements as raw SPICE elements scaled by an integer multiplicity knob, so no model card is included, but the family must be declarable for knob validation to accept it.

Family	$W_{\min}$	$W_{\max}$	$L_{\min}$	$L_{\max}$
<code>NFET_STD</code>	3.60e-7	1.0e-4	1.50e-7	2.0e-5
<code>PFET_STD</code>	4.20e-7	1.0e-4	1.50e-7	2.0e-5
<code>NFET_LVT</code>	4.20e-7	1.0e-4	1.50e-7	8.0e-6
<code>PFET_LVT</code>	4.20e-7	1.0e-4	3.50e-7	2.0e-5
<code>PFET_HVT</code>	3.60e-7	1.0e-4	1.50e-7	2.0e-5

Table 8. SKY130 1.8 V geometry envelopes from the manifest (metres).

These envelopes are a fast sanity check, not the real constraint. SKY130 device geometries are *discrete*: each device is characterized only at specific (W, L) bins. The authoritative catalog lives in `pdk/sky130/fets.json`, which records for every device its family, supply, multiplier parameter, instance prefix, pin order, default bulk, bin count, W and L ranges in both micrometres and metres, the sorted unique W and L values, and the explicit list of valid (W, L) bin pairs. Bias-current knobs are separately bounded to $[10^{-12}, 10^{-2}]$ A: a picoamp floor below which a device is effectively off, and a 10 mA ceiling above which the power budget is gone and electromigration becomes a concern.

The model library is a purpose-built reduction. `sky130.lib.spice` as shipped pulls in 5 V native-threshold devices declared with Spectre-syntax includes that ngspice cannot parse, plus ESD structures, 3.3 V/5 V families, inductors, and BJTs that no circuit here uses. We therefore build `models/sky130_1v8_only.lib.spice` containing only `nfet_01v8`, `pfet_01v8`, `pfet_01v8_hvt`, `pfet_01v8_lvt`, `nfet_01v8_lvt`, their mismatch parameters, typical R and C models, the MIM capacitor model used for compensation, and the global LOD parameters. Five corners are retained (`tt`, `ss`, `ff`, `sf`, `fs`); the campaign ran at `tt`.

5.2 Rendering and invocation

Before each simulation, `prepare_sim_workdir()` materializes a rendered copy of the testbench and topology, plus a generated `models.lib`, into the circuit's `_pdk_sim_cache/` directory, substituting `{ {INST:family} }` and `{ {MODEL:family} }` against the active manifest; ngspice then runs with its working directory set to that cache. The authored sources are never mutated, so a run leaves no derived artifacts in version control and the same source can be rendered against a different PDK without cleanup.

Invocation is `ngspice -b -n testbench` in batch mode with no init file, through a subprocess with a 180 s default timeout. Both streams are captured and concatenated, because some ngspice builds split measurement output across stdout and stderr.

Parsing is deliberately schema-free: any line of the form `name = float` is harvested into a raw measurement dictionary, so adding a new *raw* metric requires only a new `.meas` line in a testbench and no code change. Two further patterns are matched purely for diagnostics: a broad `name =` pattern catching keys whose value was unparseable (e.g. `t_rise_hi = FAILED`), and one that recovers the measure name from ngspice's failure diagnostics, which are not of `name = value` form at all. Without the third pattern, a `.meas` statement whose bracket missed the signal would be indistinguishable from an analysis that never fired.

5.3 Derived metrics

Raw `.meas` values are then extended by a registry of derived quantities. Each entry declares the raw keys it depends on and a compute function; a derivation is attempted only when every dependency is present, and failed derivations are dropped silently rather than raising. Adding a derived metric is one table row. Table 9 lists the registry.

Metric	Derivation	Metric	Derivation
<code>i_supply_a</code>	$ i_{\text{supply}} $	<code>iq_ua</code>	$ i_{\text{ground}} \cdot 10^6$

Metric	Derivation	Metric	Derivation
power_mw	$ i_{\text{supply}} V_{\text{supply}}10^3$	line_reg_mv	$ \Delta V_{\text{out}}(V_{\text{in}}) \cdot 10^3$
phase_at_ugbw_deg	rad $\rightarrow$ deg	load_reg_mv	$ \Delta V_{\text{out}}(I_L) \cdot 10^3$
phase_margin_deg	$\text{wrap}(180 + \phi_{\text{UGBW}})$	dropout_mv	$(V_{\text{in,do}} - 0.95V_{\text{nom}})10^3$
cmrr_db_100/1k	$A_{\text{OL,dB}} - A_{\text{cm,dB}}$	psrr_db_10k	$-A_{\text{supply,dB}}$
psrr_db_100/1k	$-A_{\text{supply,dB}}$	ldo_undershoot_mv	$(V_{\text{set}} - V_{\text{min}})10^3$
slew_pos/neg	$ \Delta V/\Delta t \cdot 10^{-6}$, 20–80%	ldo_settle_us	$t_{\text{settle}} \cdot 10^6$
settle_time_us	$t_{\text{settle}} \cdot 10^6$		
vout_swing_high_mv	$(V_{\text{DD}} - V_{\text{out,max}})10^3$		
vout_swing_low_mv	$(V_{\text{out,min}} - V_{\text{SS}})10^3$		

Table 9. The derived-metric registry. `ctx` supplies `supply_v`, `VDD`, `VSS`, `VOUT_NOM`, `slew_span_v` from the testbench `.param` block.

The phase-margin wrap is not cosmetic. A multi-pole loop can return a raw phase that has already wrapped past $\pm 180^\circ$, so the naive $180 + \phi$ lands at, say, 358° and an unstable circuit reads as exceptionally stable. Folding into $(-180, 180]$ makes the scorer see a small or negative margin, which is the physical truth.

A representative AnalogGym testbench runs the whole characterization in one batch job with two instances of the device under test. One is opened up (via a 10^{15} H DC-short trick) to extract open-loop gain, unity-gain bandwidth, phase margin, gain at fixed frequencies, input offset, input bias currents, quiescent current, and CMRR through AC-coupled common-mode injection. The other is closed in unity gain for the transient analyses (slew rate and settling) and the DC sweep for output swing. Stimuli are reconfigured between analyses inside the `.control` block, so a single run fires `.op`, three `.ac` sweeps, a `.tran`, and a `.dc`.

5.4 Diagnosing missing measurements

A metric can be absent for reasons that differ enormously in significance, and conflating them corrupts the learning signal: a good move whose slew extraction missed its bracket would otherwise score as catastrophically as one that destroyed the operating point. A classifier therefore assigns every missing metric one of five reason codes (Table 10), which the scorer consumes.

Code	Condition	Penalty	Passes?
<code>sim_error</code>	<code>ngspice</code> returned non-zero; the whole run failed	0.05	yes
<code>not_emitted</code>	No upstream <code>.meas</code> key appeared; the analysis never fired	0.05	yes
<code>extraction_gap</code>	All <code>.meas</code> attempted, some produced values, at least one found no crossing	0.05	yes
<code>upstream_missing</code>	Some but not all upstream keys emitted	0.50	no
<code>computation</code>	All raw deps valid but the derivation returned <code>None</code>	0.50	no
<code>unclassified</code>	No diagnostic available	1.00	no

Table 10. Missing-measurement reason codes and their scoring consequence.

The three neutral codes are tooling artifacts: they carry a small non-zero floor so the miss stays visible when comparing steps, but do not block convergence. The two partial codes indicate the metric was genuinely uncomputable.

06

The objective function

Scoring maps a measurement dictionary and a goal block to a per-target decomposition, a boolean convergence flag, and a scalar progress signal. For each target with bound b and measured value v , the raw gap is domain-aware. Metrics whose name contains `_db` are treated as logarithmic, since their violation is already expressed in decibels:

$$\text{raw} = \frac{\text{viol}_{\text{dB}}}{20}, \quad \text{versus} \quad \text{raw} = \frac{|\text{viol}|}{|b|} \quad (|b| = 0 \rightarrow 1)$$

for linear metrics. The raw gap is then soft-saturated,

$$p = \frac{\text{raw}}{1 + \text{raw}} \in [0, 1),$$

and the scalar objective is the priority-weighted sum $\text{penalty_total} = \sum_i w_i p_i$ with w_i from `goal_weights`.

Three properties of this construction are load-bearing for trace quality. Saturation prevents a metric with a small-magnitude bound (a 20 mV swing specification, say) from dominating a structurally larger failure such as a 54 dB gain deficit. Monotonicity preserves a usable gradient everywhere, so a move that improves a failing metric without fixing it still registers. And because the saturated penalty is strictly below 1.0 while the unclassified-missing sentinel is exactly 1.0, a measured-but-failing metric always scores better than one that vanished: the correct ordering, since a number you can see is more useful than one you cannot.

Saturation

A small-magnitude bound cannot dominate a structurally larger failure.

Monotonicity

A usable gradient everywhere.

Correct ordering

Measured-but-failing always scores better than vanished.

One override exists. Two metrics have a physics-level hard floor (`phase_margin_deg` and `gain_margin_db`, both at 0), below which the failure mode changes in kind rather than degree: the circuit oscillates or becomes conditionally unstable. Below the floor the penalty switches to unbounded linear growth, $p = p_{\text{soft}}(\text{floor}) + 10 (\text{floor} - v)/|b|$, continuous at the boundary so gradients survive but making an oscillating circuit unambiguously catastrophic. The override is skipped when the bound itself sits below the floor, as happens for auto-synthesized AnalogGenie targets derived from oscillating baselines.

CONVERGENCE

Convergence is decided by the conjunction of the per-target booleans, never by `penalty_total`, which is rounded for display and could otherwise round a residual failure to zero.

07

The trace-generating pipeline

7.1 Structure

A run is orchestrated by a runner that freezes a static circuit context, copies the circuit into a temporary workspace, measures a baseline, writes the header and baseline rows, then hands control to the optimization loop. The static context (assembled once per run and used as the cacheable system prefix) holds the PDK description, the knob table with families, kinds, defaults and bounds, the topology source, a stripped testbench, a metric glossary covering every metric the corpus uses (open-loop gain through comparator hysteresis and LDO dropout), the circuit's `physics_notes`, and optional retrieved physics reference material.

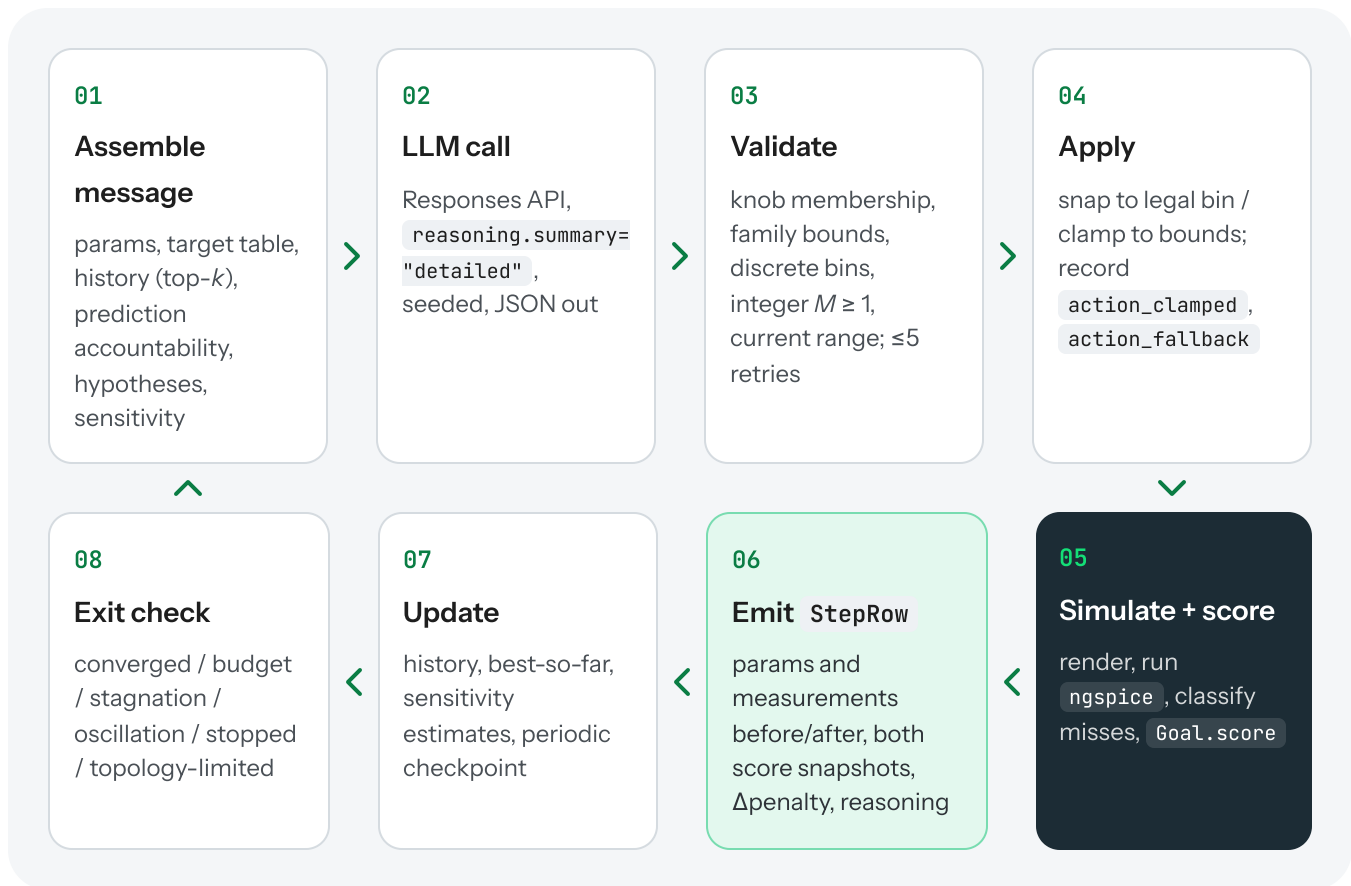


Figure 1. One iteration of the optimization loop. Every arrow crossing into stage 5 is a real `ngspice` run; every stage-6 emission is one JSONL row in the dataset.

7.2 What the model is asked, and what it must return

The per-step user message is deterministic given the trace so far. Beyond the current parameter vector and the target table, three constructed sections exist specifically to keep the recorded reasoning honest.

Prediction accountability. Each step the model must emit numeric `predictions` for the metrics it expects to move. The next step's message opens with a table of predicted versus actual values and a per-metric directional verdict; when directional accuracy falls below 50%, an explicit warning fires telling the model its causal hypothesis was wrong. The measured accuracy is stored as `prediction_directional_accuracy`, making reasoning quality a first-class, queryable field rather than something a consumer must infer.

Delta framing. Every history line carries an explicit signed Δ for each improvement and regression, so the model is never asked to do arithmetic it can get wrong.

Hypothesis falsification. Active hypotheses are listed by name with an instruction to move any contradicted by simulation data into `falsified_hypotheses`, stopping a wrong belief from propagating silently across twenty steps, and leaving a record of belief revision in the trace.

The model returns a structured object: `predictions`, a `strategy_update` (regime $\in$ {explore, exploit, diagnose}, focus metrics, active and falsified hypotheses, notes), the `action` as a knob $\rightarrow$ value map, a `decision` $\in$ {continue, hold, stop, revert}, per-parameter `action_reasoning` (a mechanism claim plus predicted direction per metric), and optional `topology_limitations`. An empty action is legal only for hold, stop, or revert.



Two are notable as trace features. `revert` lets the model recognize its last move was harmful and restore the best checkpoint; the restoration is re-simulated and written as a step row with `action_fallback = "revert"`, so reverts are visible rather than hidden. `topology_limitations` lets the model argue, with logged evidence, that a specification cannot be met by sizing at all; the loop then stops penalizing that metric and continues, and if every failing spec is so declared the run exits `topology_limited`. That distinguishes "the agent failed" from "the circuit cannot do this."

7.3 Legality enforcement

A proposed action passes a validation gate before it touches the testbench. Knobs must exist; W and L must lie inside the family envelope and any tighter per-knob bound from `circuit.json`, and coincide with a discrete PDK bin; M knobs must be integers ≥ 1 ; current knobs must lie in the manifest current range. Violations are returned to the model for repair, up to five retries per step. Only as a last resort does the applier snap to the nearest legal bin or clamp to bounds, and when it does the step row records `action_clamped` and a fallback tag (`bin_snap`, `scalar_clamp`), which downstream training filters use to exclude steps whose recorded action is not what the model asked for.

VALIDATION GATE

Knobs must exist

W, L inside envelope,
on a discrete PDK
bin

M knobs integers ≥ 1

Current knobs in
manifest range

Violation returned for repair



Up to five retries per step



Last resort: snap or clamp, tagged

7.4 Termination

Table 11 lists the exit conditions. Two are worth reading closely because they were tuned against observed failure modes.

Reason	Condition
converged	All per-target booleans true (strict mode, $\epsilon \leq 0$), or <code>penalty_total</code> $\leq \epsilon$ in loose mode.
budget	<code>max_steps</code> reached (20 in round 1, 40 in round 2).
stagnation	Over a window of 5 transitions, max – min of the penalty trajectory is below 1% of the true baseline penalty.
oscillation	Single-knob ping-pong over 4 steps, or alternating penalty deltas over 4 steps, subject to two guards (below).
stopped	Model-initiated hold or stop; current point judged best.
topology_limited	Every failing spec declared topology-limited.
parse_error	Model output could not be parsed into the schema after retries.

Table 11. Exit conditions and their parameters.

Alternating penalty deltas alone are not treated as oscillation, because a healthy try/fail/revert/try sequence also alternates. Two guards apply: a window whose deltas sum negative is making net progress and is left alone, and a window dominated by one large move (largest magnitude exceeding three times the mean of the others) is a single bad

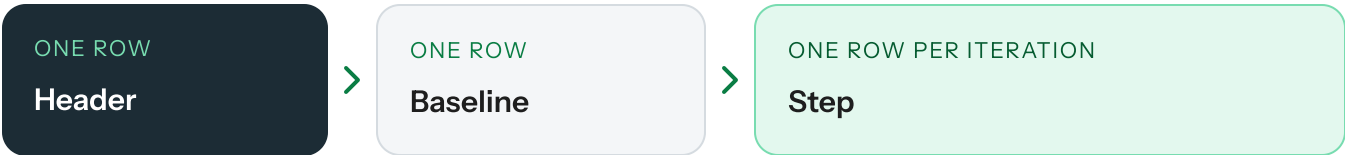
probe the model will recover from, not a cycle. Similarly, knob ping-pong is only recognized when every step in the window moves exactly one knob; in a broad exploratory bundle a knob may reverse while coupled knobs change, which is coarse-region probing.

An empirical sensitivity table runs alongside. When a step moves exactly one knob, the finite difference $\Delta\text{metric}/\Delta\text{knob}$ is recorded for every metric that moved, averaged over a 3-observation window, and injected into subsequent prompts. Entries unobserved for 10 steps are flagged stale rather than deleted, so the model can see that an estimate is old.

08

Trace format

A trace is a JSONL file at `traces/circuit.phase.jsonl` with exactly three row types. One header row, one baseline row, and one step row per iteration. The header is rewritten in place at the end of the run to attach quality metadata, so the file is self-describing after the fact.



Field	Content
<code>run_id</code> , <code>seed</code>	Unique per phase; seed passed to the API for reproducibility.
<code>circuit_name</code> , <code>circuit_family</code>	Identity.
<code>collection_phase</code>	<code>phase1</code> or <code>phase2</code> .
<code>targets</code>	The bounds this phase optimized against (possibly relaxed or finalized).
<code>original_targets</code>	The datasheet-class bounds, always retained.
<code>circuit_context</code>	The full static context: PDK, knob table, topology, stripped testbench, metric glossary, physics chunks, strategy tags.
<code>run_config</code>	Every configuration value in force.

Field	Content
<code>quality_metadata</code>	Attached on finalize: relaxation magnitude, step efficiency, prediction calibration, converged, steps used, exit reason.

Table 12. Header row.

Field	Content
<code>step_number</code>	1-indexed; baseline is step 0.
<code>params_before/after</code>	Full knob vectors, not just the delta.
<code>measurements_before/after</code>	Full metric dictionaries including nulls.
<code>score_before/after</code>	<code>penalty_total</code> plus per-target results: value, pass, status, reason, bounds, penalty, weight, weighted penalty, domain, gap.
<code>predictions</code>	Model's numeric forecast for this step.
<code>prediction_errors</code>	Per-metric error against the realized measurement.
<code>prediction_directional_accuracy</code>	Fraction of predicted directions correct; <code>null</code> when nothing moved meaningfully (distinct from 0.0).
<code>strategy_update</code>	Regime, focus metrics, active and falsified hypotheses.
<code>action</code>	Knob → value.
<code>decision</code>	<code>continue</code> / <code>hold</code> / <code>stop</code> / <code>revert</code> .
<code>action_reasoning</code>	Per-parameter mechanism claim and predicted metric directions.
<code>topology_limitations</code>	Declared unreachable metrics with evidence.
<code>action_clamped</code> , <code>action_fallback</code>	Whether the applied action differs from the requested one, and how.
<code>delta_penalty</code>	Signed change in <code>penalty_total</code> .
<code>importance</code>	Retention weight for history compression.

Field	Content
<code>retry_count</code>	Validation repair rounds consumed.
<code>reasoning_summary</code>	Native reasoning summary from the Responses API.
<code>duration_s</code>	Wall-clock for the iteration (LLM + sim + scoring).
<code>effective_supply_v</code>	Live supply, which may itself be a knob.

Table 13. Step row. The baseline row is the same shape without the action and prediction fields.

Reasoning capture deserves a note on method. Raw chain-of-thought is never exposed by the provider. An earlier implementation asked the model to emit a `reasoning_chain` field in its JSON, which made it re-narrate a rationale as billed output tokens, paying twice, and yielding a post-hoc rationalization rather than a record of the actual process. The current pipeline uses the Responses API with `reasoning.summary = "detailed"`, returning a summary of the genuine internal reasoning as a separate output item: more faithful and cheaper, since it rides on tokens already billed. Each summary is stored in the step row and injected into the next step's prompt under *"Your reasoning from the previous step"*.

09

The two-round collection campaign

9.1 Why two rounds

Optimizing against a datasheet specification produces almost no converged traces, and a corpus of uniformly failing episodes teaches very little. The reason is not that the optimizer is weak but that the constraint is wrong: a recycling-amplifier topology asked for 100 dB of open-loop gain and 68 dB of CMRR on a 130 nm process at 1.8 V may have no legal sizing that satisfies it. What is needed is a target set that is demanding but reachable, and the only honest way to find that boundary is to measure it. The campaign therefore runs in two rounds: round 1 discovers what each circuit can do, a derivation step converts that evidence into a finalized target set, and round 2 re-optimizes against those targets with a larger budget and no relaxation. Both rounds are retained: round 1 traces are episodes of grappling with an infeasible spec, itself a useful behaviour to learn from.

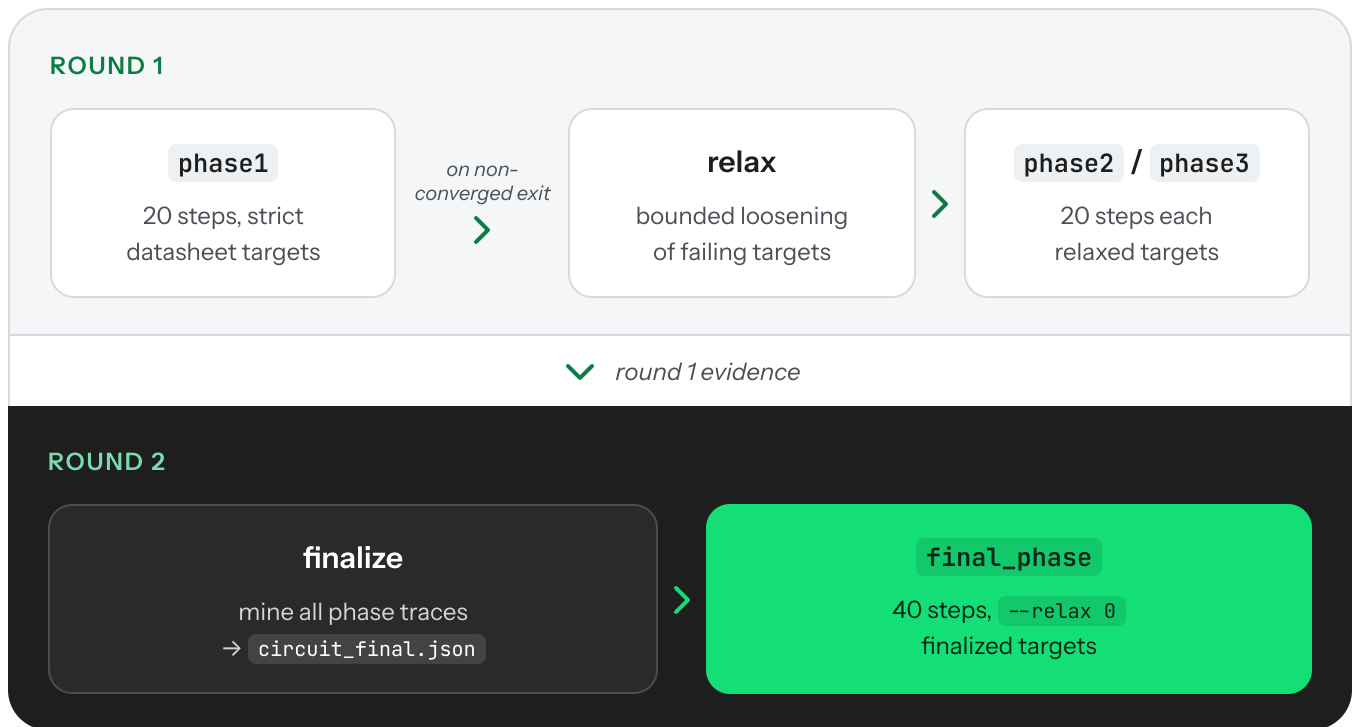


Figure 2. The two-round protocol. Round 1 is the `phase1` → `relax` → `phase2` → ... chain; round 2 is `final_phase` .

9.2 Round 1: relaxation chain

Round 1 runs at a 20-step budget per phase with one relaxation round enabled. Circuits are discovered by scanning the configured netlists directory for `circuit.json` files, so a set is defined by its directory rather than by a hard-coded list. Each circuit runs in its own thread with its own API client, so there is no shared mutable state; the worker count is configurable and defaults to one worker per circuit.

Within a circuit, `phase1` optimizes against the original targets; on any non-converged exit a relaxation step computes new bounds and a further phase runs against them. `--relax 1` yields the `phase1` → `phase2` pair; a second round produces `phase3` . Each relaxed phase writes its own trace file, so no evidence is overwritten.

The relaxation itself is deterministic: no language model, no simulation. It keys off the measurements at the *best point* of the previous phase, not off whether a metric ever passed at some worse-overall step, because a metric that passed transiently but fails at the convergence candidate must still be relaxed or the chain dead-ends. For a lower-bounded target with bound $b_{\min}$ and achieved value a :

$$b'_{\min} = \min(b_{\min}, \max(a - m, b_{\min} - c_{\text{dB}})) \quad (\text{logarithmic metrics, } c_{\text{dB}} = 20.0)$$

$$b'_{\min} = \min(b_{\min}, \max(a(1 - m), b_{\min} - c_{\text{lin}} |b_{\min}|)) \quad (\text{linear metrics, } c_{\text{lin}} = 0.30)$$

with the symmetric form for upper-bounded targets and margin $m = 0$ in the campaign. Three properties follow. The outer min guarantees a bound is never tightened, so a metric already passing at the best point no-ops. The cap (at most 20 dB per round for decibel metrics, at most 30% of the bound magnitude for linear ones) means relaxation clamps rather than capitulating, and is applied per round against the previous round's bound, so successive rounds compound. And a metric that was never measured keeps its original bound, letting the next phase try again instead of aborting the chain. The chain is declared infeasible only when nothing at all could be relaxed.

9.3 Deriving achievable targets

Between the rounds, a target-finalization step converts round-1 evidence into a finalized goal block. It scans every row of every `phase*` trace for a circuit and selects the single best point (most original targets passing, ties broken by lowest `penalty_total`), then assigns each metric a bound by one of four rules:

Provenance tag	Rule
<code>passed_bound</code>	Metric passed at the best point: keep the bound it met, giving round 2 margin.
<code>achieved_at_best</code>	Metric failed at the best point: relax the original bound to the value measured there.
<code>best_achieved_elsewhere</code>	Not measured at the best point: use the value from any row with the smallest violation of the original bound, the minimal relaxation consistent with the evidence.
<code>kept_original_never_measured</code>	Never measured anywhere: keep the original bound.

Table 14. Target-derivation rules applied at finalization.

Relaxation is one-sided where the target is two-sided: only the breached side of a `{min, max}` bound moves. The output, `circuit_final.json`, carries both the new goal and a provenance block recording the source phase and step of the best point, how many targets

passed there, and per metric the rule applied, the original bound, the new bound, and the value used. Context construction auto-prefers this file, so round 2 needs no flag to pick it up; the run log confirms it, e.g. " ag_comparator_papers_1058: using circuit_final.json goal targets. "

For Alfio_RAFFC_Pin_3 the best point was phase1 step 20, with 9 of 17 targets passing at penalty_total = 3.708. Table 15 shows the eight targets that moved; nine held their original bounds because they had already passed.

Metric	Original	Derived	Rule
gain_db	≥ 100	≥ 55.34	achieved_at_best
cmrr_db_100	≥ 68	≥ 35.97	achieved_at_best
cmrr_db_1k	≥ 68	≥ 35.97	achieved_at_best
psrr_db_100	≥ 80	≥ 36.89	achieved_at_best
psrr_db_1k	≥ 80	≥ 36.89	achieved_at_best
slew_pos_v_per_us	≥ 2.0	≥ 0.551	achieved_at_best
vout_swing_high_mv	≤ 20	≤ 765.3	achieved_at_best
vout_swing_low_mv	≤ 20	≤ 519.1	achieved_at_best

Table 15. Derived targets for Alfio_RAFFC_Pin_3 . Unchanged targets (ugbw_hz , i_supply_a , power_mw , phase_margin_deg , vos_uv , ib_pos_pa , ib_neg_pa , settle_time_us , slew_neg_v_per_us) are omitted.

The magnitude of these moves is the honest finding of round 1, and it is worth stating plainly rather than burying: a 130 nm process at 1.8 V does not deliver TLV9001 rail-to-rail output headroom or 100 dB open-loop gain on these topologies. Recording the gap explicitly (in a provenance block, traceable to a specific simulated step) is more useful than pretending otherwise.

Of the 103 circuits, 96 received a circuit_final.json : all 20 AnalogGym and 76 of the 83 AnalogGenie circuits. The remainder had no scorable round-1 rows.

9.4 Round 2: final phase

Round 2 runs with relaxation off and double the budget: 40 steps per circuit. Traces are written to `circuit.final_phase.jsonl`. Because the targets are now demonstrably achievable, convergence becomes a meaningful event, and the larger budget gives the optimizer room to reach it.

10

Dataset statistics

10.1 Inventory

Set	Ckts	phase1	phase2	phase3	final	Traces	Rows
AnalogGym	20	20	19	15	19	73	1,406
AnalogGenie	83	83	73	n/a	75	231	4,082
Total	103	103	92	15	94	304	5,488

Table 16. Trace inventory. Row counts are header + baseline + step rows.

The 5,488 rows decompose exactly as 304 headers, 304 baselines, and 4,880 step rows, and therefore 4,880 ngspice runs beyond the 304 baseline simulations, since every step re-simulates. Only AnalogGym ran a second relaxation round, hence its `phase3` files. Where a phase count falls below the circuit count, that circuit either converged in the previous phase (so no relaxation was needed) or errored.

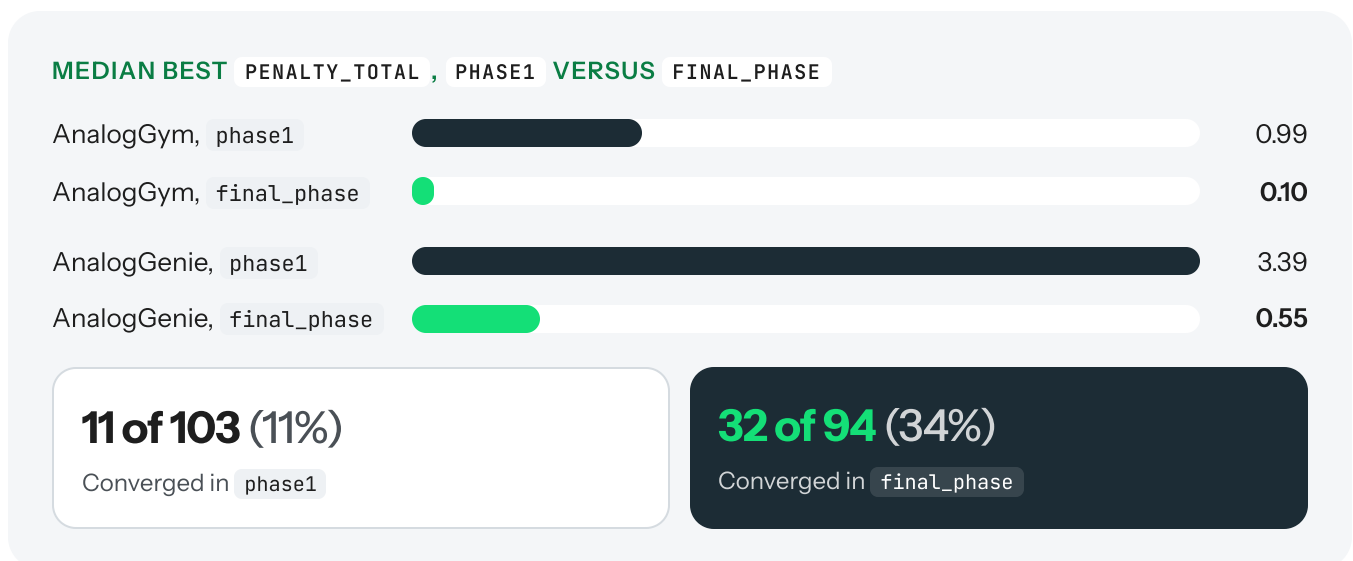
10.2 Optimization outcomes

Table 17 reports, per set and phase, median steps taken, median baseline and best `penalty_total`, median relative improvement, converged runs, and median targets met at baseline versus at the best step.

Set	Phase	n	Steps	Base	Best	Impr.	Conv.	Met
AnalogGym	phase1	20	17	7.92	0.99	85.4%	1	5 → 11 / 17
	phase2	19	18	7.77	0.56	92.1%	2	5 → 11 / 17
	phase3	15	20	7.38	1.40	81.4%	1	5 → 10 / 17
	final_phase	19	21	5.88	0.10	98.0%	7	6 → 15 / 17
AnalogGenie	phase1	83	14	8.79	3.39	62.8%	10	5 → 5 / 11
	phase2	73	19	7.85	1.94	69.0%	10	6 → 9 / 11
	final_phase	75	15	4.92	0.55	81.6%	25	6 → 11 / 11

Table 17. Measured outcomes. Penalties are `penalty_total` medians; improvement is median per-trace relative reduction.

The two-round design is validated by the `final_phase` row of each block. On AnalogGym, median best penalty falls from 0.99 in `phase1` to 0.10 in `final_phase` (roughly a tenfold reduction), converged runs rise from 1 to 7, and median targets met rise from 11 to 15 of 17. On AnalogGenie, median best penalty falls from 3.39 to 0.55, converged runs rise from 10 of 83 to 25 of 75, and median targets met rise from 5 to all 11. Aggregated across all phases, 56 of 304 runs converged; restricted to `final_phase`, 32 of 94 (34%) converged, against 11 of 103 (11%) in `phase1`.



Median baseline penalty also falls between rounds (7.92 to 5.88 on AnalogGym): the mechanical effect of scoring the same starting point against finalized rather than datasheet targets, and the reason improvement percentages should be read alongside the absolute

best penalties.

10.3 Agent behaviour

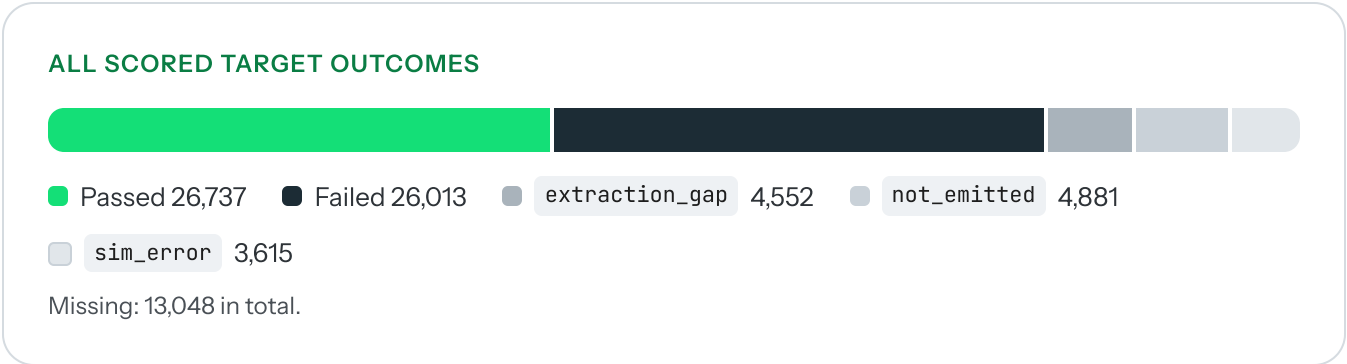
Quantity	Value
Step rows	4,880
decision <code>continue</code>	4,021 (82.4%)
decision <code>revert</code>	831 (17.0%)
decision <code>stop</code>	24
decision <code>hold</code>	4
Steps with a non-empty <code>reasoning_summary</code>	3,935 (80.6%)
Median <code>reasoning_summary</code> length	1,379 characters
Steps with a scored directional accuracy	3,927
mean directional accuracy	0.724
median directional accuracy	0.818
Steps where all measurements were null	3

Table 18. Behavioural statistics over all 4,880 step rows.

Two figures here matter for anyone using the dataset. The 17% revert rate means roughly one step in six is a recorded self-correction, making the corpus unusually rich in error-recovery behaviour and, because reverts carry an explicit `action_fallback`, easy to filter in or out. The directional accuracy distribution (mean 0.724, median 0.818) says the model's causal predictions were right about three times in four; the gap between mean and median indicates a left tail of steps where its mechanism model was simply wrong, which is arguably the most interesting supervision in the set.

Only 3 steps of 4,880 produced a wholly null measurement dictionary, so outright simulation collapse is rare. Partial misses are not: across all scored targets, 26,737 outcomes passed and 26,013 failed, while 13,048 were missing (4,552 classified `extraction_gap`, 4,881

not_emitted , and 3,615 sim_error). About a fifth of all target evaluations therefore rest on a metric that could not be extracted, which is why the diagnostic codes and the neutral penalty floor matter: a consumer computing aggregate spec satisfaction should exclude or explicitly model these rather than treating a neutral miss as a pass.



10.4 Termination

Exit reason	Count	Share	
oscillation	93	30.6%	
budget	73	24.0%	
converged	56	18.4%	
topology_limited	31	10.2%	
stopped	30	9.9%	
stagnation	18	5.9%	
parse_error	3	1.0%	
Total	304		

Table 19. Exit reasons across all 304 traces.

Oscillation is the single most common exit, consistent with what the guards in Section 7 were built to distinguish: on a 17-constraint objective with strongly coupled knobs, a policy that pushes gain, loses phase margin, recovers it, and loses gain again is the default failure

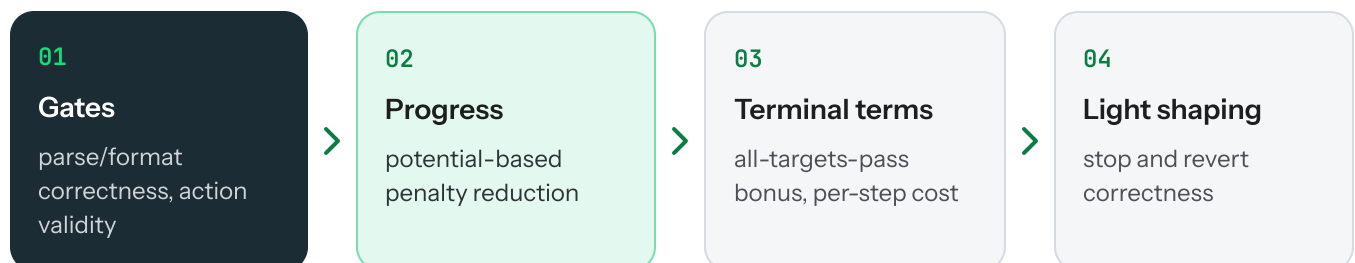
mode. The 31 `topology_limited` exits are cases where the model argued, with logged evidence, that a specification was unreachable by sizing: a claim a consumer can check against the recorded sweep.

11

Downstream use

The corpus was collected to train an analog-sizing policy, and the trace schema is shaped by that intent. For supervised fine-tuning, each retained step becomes a (system, user, assistant) triple. The system message is the shared per-step prompt including the discrete *W/L* bin catalog, and must match the reinforcement-learning rollout prompt exactly so the two stages do not disagree about what the model may do; the user message reconstructs the step context; the assistant message is the recorded reasoning in `<think>` tags followed by the action JSON. Retention filters on two axes. *Hard validity* excludes any step whose recorded action was not legally what the model asked for: off-bin *W/L*, non-integer or sub-unity *M*, out-of-range current, an unknown knob, or any `action_clamped` / error-fallback tag. *Quality* keeps steps that improved the score immediately or produced a positive discounted forward return, with per-step reward $-\Delta\text{penalty}$; hold, stop and revert steps are kept regardless, for coverage.

For reinforcement learning the same fields define a tiered reward: non-tradeable gates on parse/format correctness and action validity, then potential-based progress using penalty reduction with the potential saturated at the bound, then terminal terms (an all-targets-pass bonus and a small per-step cost), then light shaping on stop and revert correctness, with group-relative advantages across the completions sampled per prompt. The environment hooks are the two functions the collection pipeline used: run `ngspice`, then score with `Goal.score`, so the reward at training time is computed by the code that produced the labels, removing an entire class of train/serve skew.



Conclusion

The artifact described here is a corpus of 304 complete, simulator-adjudicated analog optimization episodes over 103 SKY130 circuits: 4,880 steps, every measurement produced by a real ngspice run, every score decomposable to its per-target constituents, and every step carrying the model's predictions, hypotheses, and reasoning alongside its action.

Three design decisions carried the pipeline. Separating the PDK-neutral topology from the PDK manifest turned a hundred heterogeneous circuits into one uniform, retargetable representation: a single code path renders, simulates and scores an AnalogGym multi-stage amplifier and a 113-knob AnalogGenie regulator alike. Domain-aware, soft-saturated, diagnostic-adjusted scoring gave the optimizer a progress signal that stays monotonic and informative even when a measurement fails to extract, so partial instrumentation degrades the signal gracefully instead of poisoning it. And the two-round protocol (bounded deterministic relaxation then evidence-derived target finalization) converted an unreachable specification into a demanding but attainable one without inventing a number: every finalized bound is a value the circuit was measured to reach.

DECISION 1

PDK-neutral topology
separated from the PDK
manifest

DECISION 2

Domain-aware, soft-
saturated, diagnostic-
adjusted scoring

DECISION 3

The two-round protocol

The measured outcome is a pipeline that improves circuits reliably and at scale. Median relative penalty reduction runs 81 to 98% across the AnalogGym phases and 63 to 82% across the AnalogGenie phases, reaching 98.0% and 81.6% in the final phase. Convergence rises from 11% of round-1 runs to 34% in round 2; median best penalty falls tenfold on AnalogGym (0.99 to 0.10) and sixfold on AnalogGenie (3.39 to 0.55); and median targets simultaneously met rises from 11 to 15 of 17 on AnalogGym and from 5 to all 11 on AnalogGenie, on circuits whose canonical starting point frequently does not even bias. Across 4,880 steps, outright simulation collapse occurred 3 times, causal predictions were directionally correct with a median accuracy of 0.818, and 17% of steps were self-corrections the loop detected, reverted and recovered from unaided.

Just as important, the pipeline is auditable end to end: every trace retains the original datasheet targets beside the ones optimized, every finalized bound carries its provenance, every clamped action is tagged, every termination is labelled with its triggering condition, and every claim in this paper can be recomputed from the shipped JSONL.

WITHOUT INVENTING A NUMBER

**Every finalized bound is a value
the circuit was measured to reach.**

References

1. Texas Instruments. *TLV900x Low-Power, RRIO, 1-MHz Operational Amplifier for Cost-Sensitive Systems*. Datasheet SBOS703 (rev. SBOS833R). Goal anchor for the AnalogGym amplifiers and the AnalogGenie amplifier/opamp families. <https://www.ti.com/lit/ds/symlink/tlv9001.pdf>
2. Texas Instruments. *TLV906x 10-MHz, RRIO, CMOS Operational Amplifier*. Datasheet SBOS770. Anchor for Leung_NMCF_Pin_3 (100 pF load). <https://www.ti.com/lit/ds/symlink/tlv9061.pdf>
3. Texas Instruments. *TPS7A02 200-mA, Nanopower- I_Q , Low-Dropout Voltage Regulator*. Datasheet SBVS277. Anchor for the LDO family. <https://www.ti.com/lit/ds/symlink/tps7a02.pdf>
4. Texas Instruments. *TLV3201, TLV3202 40-ns, Rail-to-Rail I/O Comparator*. Datasheet SBOS589. Anchor for the comparator family. <https://www.ti.com/lit/ds/symlink/tlv3201.pdf>
5. Texas Instruments. *LM4040 Precision Micropower Shunt Voltage Reference*. Class anchor for the reference family. <https://www.ti.com/lit/ds/symlink/lm4040.pdf>
6. AnalogGym: An Open and Practical Testing Suite for Analog Circuit Synthesis. ICCAD 2024, arXiv:2409.08534. Source of the 20 hand-curated circuits. <https://github.com/CODA-Team/AnalogGym>
7. AnalogGenie: A Generative Engine for Automatic Discovery of Analog Circuit Topologies. ICLR 2025, arXiv:2503.00205. Source of the 83 converted circuits. <https://github.com/xz-group/AnalogGenie>
8. SkyWater Technology and Google. *SkyWater SKY130 Open Source PDK*. <https://skywater-pdk.readthedocs.io/> <https://github.com/google/skywater-pdk>
9. ngspice project. *Ngspice, the Open Source SPICE Circuit Simulator: User Manual*. <https://ngspice.sourceforge.io/docs.html>
10. B. Razavi. *Design of Analog CMOS Integrated Circuits*, 2nd ed. McGraw-Hill, 2017.
11. B. Razavi. *RF Microelectronics*, 2nd ed. Prentice Hall, 2011.

12. H. Camenzind. *Designing Analog Chips*, 2005.
13. T. C. Carusone, D. Johns and K. Martin. *Analog Integrated Circuit Design*, 2nd ed. Wiley, 2011.
14. P. R. Gray, P. J. Hurst, S. H. Lewis and R. G. Meyer. *Analysis and Design of Analog Integrated Circuits*, 5th ed. Wiley, 2009.
15. P. E. Allen and D. R. Holberg. *CMOS Analog Circuit Design*, 3rd ed. Oxford University Press, 2011.