AICHIPBENCH: STATE-AWARE EVALUATION OF CODING AGENTS FOR AI ACCELERATORS

Anonymous authors
Paper under double-blind review

ABSTRACT

An RTL accelerator can emit the correct token while corrupting the state that later tokens depend on. For a coding agent, such delayed failures are hard to debug: the first visible mismatch may appear well after the faulty update. We introduce AIChipBench to evaluate whether agents can use public verification feedback to build implementations that pass independent acceptance as cross-module integration and persistent-state requirements increase. Each task pairs an executable fixed-point contract with a public validation ladder, an isolated held-out evaluator, and tool-use accounting, so that model fidelity, implementation correctness, and agent reliability are measured separately. On three controlled integration tasks with a four-layer Qwen-derived test model (Qwen3.5-nano), each attempted 15 times (5 attempts from each of 3 agent configurations), final acceptance is 11/15 for a single layer, 7/15 for a four-layer one-step decoder, and 2/15 for eight-step recurrent decoding; all nine attempts that pass the public ladder but are rejected fail the held-out numerical checks. Scaffold-adaptation case studies cover Qwen3.5-0.8B and 4B; three Qwen reference designs satisfy their finite suites, while additional Gemma, Llama, and Qwen3.6-27B checks probe the limits of scaffold reuse. With arithmetic held fixed, paired schedules improve decode throughput by 12.5–15.0% at essentially unchanged LUT usage. Our benchmark separates the feasibility of a hardware design from an agent's ability to construct it reliably, making state-preserving integration a concrete target for agent evaluation.

1 INTRODUCTION

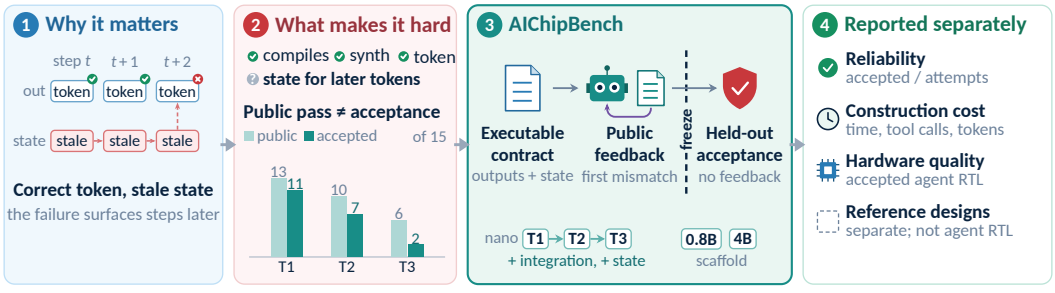


Figure 1: AIChipBench overview. (1) Schematic: an accelerator can emit a correct token while carrying stale state that corrupts a later step. (2) Compilation, synthesis and current-token checks do not establish that state is preserved, and public-ladder success overstates final acceptance (pilot counts out of 15 attempts per task; Table 1). (3) Each task pairs an executable integer oracle for outputs and state, whose fidelity to FP32 is checked separately, with public feedback and, after the freeze, isolated held-out acceptance. Nano is Qwen3.5-nano, our compact test model rather than an official Qwen release; its pilot tasks T1–T3 (one layer, a four-layer decoder, eight decode steps) add integration and recurrent state, while 0.8B and 4B adapt a shared scaffold. (4) Reliability, construction cost and hardware quality (cycles and FPGA LUTs for accepted agent RTL) are reported separately; reference designs are the paired schedules of Section 6, not agent submissions. Figure 4 details the protocol.

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

Consider an autoregressive accelerator that computes the current token correctly but fails to commit an updated convolution buffer. The output check passes, yet the next decoding step consumes stale state. When a token mismatch finally appears, its cause may lie several module boundaries or transactions earlier; a reduced replay is shown in Figure 3a and Appendix I. A coding agent must trace that delayed failure back to the faulty update, repair the interacting modules, and preserve behavior that already passed. Compilation, synthesis, and agreement on the current token all provide useful feedback, but none of them alone establishes that the implementation preserves the state that later outputs need.

We therefore ask: *As cross-module integration and persistent-state obligations increase, can coding agents use public verification feedback to construct implementations that pass independent final acceptance?* Existing RTL benchmarks generally evaluate functional correctness and hardware quality across many designs, and agentic benchmarks add multi-turn design and verification (Lu et al., 2024; Garcia-Gasulla et al., 2025; Pinckney et al., 2025). Figure 2 contrasts these evaluations with ours, which varies integration obligations within a single family of Qwen-derived text decoders. RTL for AI accelerators is a precise setting for this question: an executable reference specifies both numerical outputs and persistent state, so an isolated layer can be extended into a complete decoder and then into recurrent decoding while the constituent arithmetic stays fixed. Errors in intermediate state then become directly observable, even before they change the predicted token.

We introduce AIChipBench to study this setting. Each task specifies a fixed-point state-transition contract, a public validation ladder, and an isolated held-out evaluator (Figures 1 and 4). During construction, the public ladder returns localized feedback on outputs and state to guide debugging. After the agent exits, its submission is frozen and evaluated on new inputs, state histories, resets, and bounded memory stalls. Construction time and tool use are recorded separately from the hardware characteristics of the resulting design. The protocol therefore distinguishes three outcomes: (1) whether the numerical representation remains faithful to the original model, (2) whether the RTL implements that representation bit-exactly, and (3) whether an agent can reliably construct an accepted implementation within a fixed budget.

The paper makes three contributions. Sections 3 and 4 define an evaluation contract that separates numerical approximation from bit-exact RTL implementation. Section 5 uses this contract to study how final acceptance under public feedback changes from single-layer construction to cross-layer integration and stateful decoding. Section 6 compares paired schedules to show what scheduling can improve once arithmetic correctness is fixed. Table 1 collects the agent and hardware results, and Figure 3 adds two diagnostics. We study these questions on Qwen-derived text decoders, with short held-out tests within the declared interface. Across fresh nano-task attempts, acceptance falls from 11/15 at the single-layer level to 2/15 under recurrent decoding. Gate-level analysis places the public-to-final gap in the held-out numerical checks, and a reduced transition replay shows how a faulty state update can precede a visible output error. All three Qwen reference designs pass their declared RTL suites, which addresses task feasibility; the 0.8B and 4B case studies examine adaptation of a shared scaffold, not independent discovery of an accelerator architecture. Our conclusions are bounded by the evaluated agents, interfaces, and decoding horizons.

2 Related Work

RTL generation and evaluation. RTL generation has been evaluated at several levels. HDLBits (Wong, 2024) provides automatically checked HDL exercises, and VerilogEval (Liu et al., 2023b) uses such problems to evaluate language models through functional simulation. RTLLM (Lu et al., 2024) and its 2.0 release (Liu et al., 2024) evaluate syntax, functionality, and design quality; TuRTLe (Garcia-Gasulla et al., 2025) draws on several of these datasets and scores syntax, function, synthesis, power, performance, and area (PPA), and line completion; and RTL-Repo (Allam & Shalan, 2024) completes RTL within larger codebases (Table 3). ChipBench (Yu et al., 2026) evaluates Verilog generation on 44 modules, including hierarchical designs and CPU IP, alongside debugging and reference-model generation. AIChipBench complements that breadth with a controlled task family in which cross-module integration and persistent-state requirements vary under a shared arithmetic contract.

Work	Task emphasis	Verification emphasis	Reported outcome
RTLLM	RTL generation across designs	Syntax, function, synthesis	Correctness, design quality
RTL-Repo	Completion in codebases	Match to the reference line, given repository context	Completion quality
CVDP	Design and verification tasks	Hidden test harnesses	Task success
AIChipBench (ours)	Fixed-point decoder contracts; T1–T3 increase integration and recurrent state under one contract	Public feedback; frozen, held-out state-exact acceptance	Agent reliability, construction cost, paired hardware quality

Figure 2: AIChipBench compared with representative RTL evaluations. Prior work emphasizes functional and hardware quality across designs, repository-aware completion, or agentic design and verification (Lu et al., 2024; Garcia-Gasulla et al., 2025; Allam & Shalan, 2024; Pinckney et al., 2025). AIChipBench instead controls integration depth and persistent state under one fixed-point decoder contract (tasks T1–T3), and its outcomes include frozen, state-exact acceptance and construction cost.

Table 1: Key results. **A.** Independent Qwen3.5-nano attempts by coding-agent configuration; configurations are named by provider and do not form a controlled comparison of base-model weights. Every count is out of all attempts in that cell: *public pass* means the attempt passed the public ladder, and *accepted* means it passed every final gate. Timeouts (6 attempts, all Kimi system) count as failures. Every public-pass rejection failed the held-out numerical checks and passed synthesis and integrity (all nine are listed in Table 10). The pooled row sums all 15 attempts per task, five from each configuration. **B.** One baseline and one optimized reference design per workload, not agent submissions, at 126 MHz, batch size one, and an eight-token prompt, with arithmetic and hardware capacity held fixed within each pair. Efficiency is tokens per joule, and Section 6.2 gives energy per token. Δ is the change relative to the same workload's baseline; LUT counts cover the mapped accelerator, not external weight capacity. Shading: blue encodes n/N on one scale; green encodes throughput and efficiency gains on one scale; power and LUT changes are not shaded.

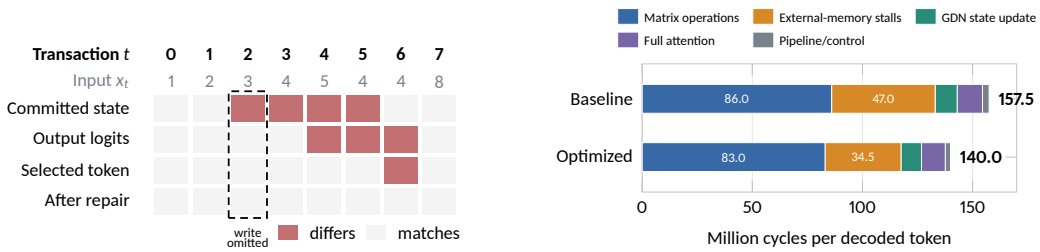
A. Coding-agent attempts (Qwen3.5-nano; counts out of all attempts)

	T1 Single layer		**T2** Four-layer, one-step decoder		**T3** Eight-token recurrent decoder	
Agent configuration	Public pass	Accepted	Public pass	Accepted	Public pass	Accepted
Codex system	5/5	4/5	4/5	3/5	3/5	1/5
Claude system	4/5	4/5	3/5	2/5	2/5	1/5
Kimi system	4/5	3/5	3/5	2/5	1/5	0/5
Pooled	**13/15**	**11/15**	**10/15**	**7/15**	**6/15**	**2/15**
Public-pass rejections	2 of 13		3 of 10		4 of 6	

B. Paired reference designs (baseline vs. optimized schedule; same arithmetic and memory model)

	Decode throughput (tokens/s)			Boundary power (W)			Energy efficiency	Mapped LUTs		
Workload	Base	Opt.	Δ	Base	Opt.	Δ	Δ	Base	Opt.	Δ
Qwen3.5-nano	365.22	420.00	+15.0%	6.0	6.2	+3.3%	+11.3%	47,100	47,050	−0.11%
Qwen3.5-0.8B	4.00	4.60	+15.0%	9.0	9.4	+4.4%	+10.1%	61,800	61,750	−0.08%
Qwen3.5-4B	0.80	0.90	+12.5%	10.0	10.5	+5.0%	+7.1%	68,980	68,920	−0.09%

Agents and verifiable learning. Hardware design also provides executable feedback for agent evaluation and learning. CVDP (Pinckney et al., 2025) includes both non-agentic and multi-turn agentic tasks spanning design and verification. The LBC-bench leaderboards (LBC-bench Team, 2025) report model and agent results on CVDP: code generation is scored by simulation pass/fail



(a) State diverges before output.

(b) Scheduling reduces memory stalls.

Figure 3: Diagnostics for the results in Table 1. (a) Reduced replay (Appendix I): omitting the accepted state write at transaction 2 makes the committed state differ from the reference at transactions 2–5, the logits at 4–6, and the selected token only at 6; restoring the write removes every mismatch. (b) For Qwen3.5-4B, the optimized schedule reduces decode cost from 157.5 to 140.0 million cycles/token, mainly through fewer external-memory stalls (47.0 to 34.5 million).

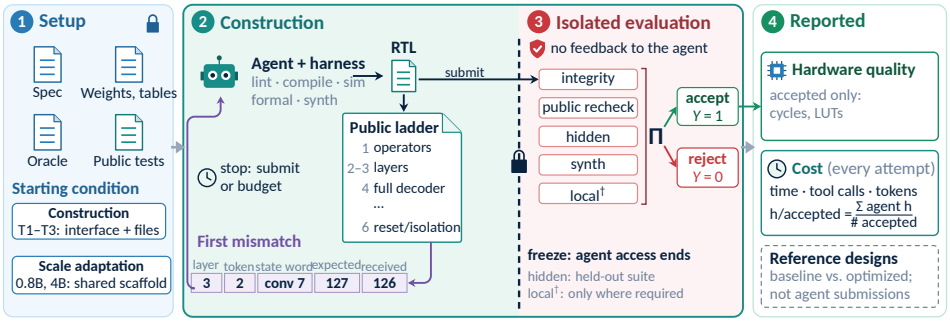


Figure 4: How a submission is built and accepted. (1) The task files are protected (tables: nonlinear lookup tables); construction tasks T1–T3 start from supplied files, and scale adaptation (0.8B, 4B) starts from a shared scaffold. (2) The harness runs lint, compilation, simulation (sim), bounded property checks (formal) and synthesis (synth). The public ladder (six stages, abbreviated; Section 3) returns first-mismatch records, here the example of Appendix D (conv 7: a convolution-state word); a run stops at submission or a budget limit. (3) At the freeze, agent access ends; the isolated evaluator returns nothing to the agent and multiplies (Π) the gates of Equation 2. (4) Cost covers every attempt; h/accepted divides total agent hours by the number accepted. Hardware quality (cycles, FPGA LUTs) covers accepted submissions only; the reference designs of Section 6 are a separate track. Appendix L also tests output-only feedback.

and pass@k, non-agentic comprehension by BLEU and LLM-based judging, and agentic heavy-context generation in open-source repositories, which emphasizes repository exploration, tool use, and integration, by repository build and testbench execution. Our tasks center on AI-accelerator RTL with persistent decoding state, varying integration depth across the three Qwen3.5-nano tasks under one fixed-point contract. Architect Labs (Architect Labs, 2026) studies single-turn RTL generation through reinforcement learning with verifiable rewards, curriculum learning, and reward shaping, and identifies agentic EDA as a further direction. At the workflow level, ChatEDA (Wu et al., 2024) combines task decomposition, script generation, and tool execution to orchestrate RTL-to-GDSII flows. ChipNeMo (Liu et al., 2023a) studies domain-adapted models for engineering assistance, EDA script generation, and bug analysis. These systems assign different responsibilities to the model and expose different feedback. We fix the evaluation protocol and measure how reliably an agent turns public feedback into accepted implementations as integration and state obligations grow.

Verification across design stages. Verification is also central to workflows that move between design representations. EDASYTH (Yuen & Hoang, 2026) connects symbolic calculations, LTspice simulation, and KiCad conversion through checks that gate progression to the next stage. Although it targets analog circuits rather than digital RTL, it shares our emphasis on checking intermediate representations before accepting a final design. The large-circuit-model perspective (Chen

et al., 2024) places these transitions within a broader agenda of learning across specifications, RTL, netlists, and physical layouts. AIChipBench addresses a specific part of that agenda: evaluating whether an agent preserves the executable semantics of an AI model when constructing RTL. It contributes an evaluation protocol for that transition, not a learned representation spanning the design flow. EEBench (atopile, 2026) also evaluates engineering agents, scoring submitted design bundles by simulation checks and bill-of-materials cost. We differ in varying the integration obligations of an LLM workload in a controlled way and in separating public feedback from state-exact final acceptance.

3 TASK CONTRACT AND EVALUATION PROTOCOL

Table 2: AIChipBench at a glance. Reference-suite exactness compares RTL with the frozen integer oracle; top-1 fidelity compares fixed point with FP32 on common prefixes. Agent cohorts and paired schedules are in Table 1. The nano workload misses the separate 99% fidelity criterion.

Unit	Scope	Validation or comparison	Reported result
Qwen3.5-nano	4 layers (3 GDN, 1 attention); width 256	Public RTL exact: 1,384/1,384	FP32 top-1: 97.22%
Qwen3.5-0.8B	24 layers (18 GDN, 6 attention); width 1,024	Public: 3,120/3,120; held-out: 232/232 exact	FP32 top-1: 99.31%
Qwen3.5-4B	32 layers (24 GDN, 8 attention); width 2,560	Public: 3,680/3,680; held-out: 288/288 exact	FP32 top-1: 99.22%

Table 3: Size and scoring of representative RTL benchmarks: VerilogEval (Liu et al., 2023b), RTLLM (Lu et al., 2024), RTLLM 2.0 (Liu et al., 2024), RTL-Repo (Allam & Shalan, 2024), TuRTLe (Garcia-Gasulla et al., 2025) and CVDP (Pinckney et al., 2025). Counts keep each source's unit. TuRTLe reuses several of the other datasets, so its counts overlap theirs. For AIChipBench (ours), the counts are agent attempts: 45 in the pilot and 152 in the Sol/Astra comparison.

Benchmark	Count	Unit	Content	Scoring
VerilogEval	156	problems	HDLBits modules, from combinational logic to state machines	Simulation against a reference
RTLLM	30	designs	Median 52 HDL lines and 121 netlist cells per design	Syntax, function, synthesized design quality
RTLLM 2.0	50	designs	Descriptions, reference RTL and test cases	Syntax, function, design quality
RTL-Repo	4,098	examples	Multi-file Verilog projects; 1,174-example test split	Exact match and edit similarity of one completed line
TuRTLe	1,397	tasks	1,174 RTL-Repo line completions and 223 problems from VeriGen, VerilogEval and RTLLM (50 designs); 40 LLMs evaluated	Syntax, function, synthesis, PPA; exact match for line completion
CVDP	783	problems	13 categories of design, verification, debugging and comprehension; agentic and non-agentic formats	Hidden test harnesses; BLEU or LLM judging for comprehension answers
AIChipBench	**45; 152**	**attempts**	**Agent tasks on Qwen3.5-nano (T1–T3) and Qwen3.5-0.8B; the optimized reference designs for three Qwen decoders use 47,050–68,920 LUTs**	**Exact outputs and state on public and held-out checks; synthesis; integrity; local checks where required**

3.1 SEPARATE APPROXIMATION FROM IMPLEMENTATION

For an input token x_t, position p_t, weights W and persistent state s_t, we define the integer oracle as

$$(z_t, s_{t+1}) = F_q(x_t, p_t, s_t; W), \qquad s_t = (s_t^{\mathrm{GDN}}, s_t^{\mathrm{conv}}, K_t, V_t, m_t), \tag{1}$$

where z_t contains the logits and m_t includes valid lengths and control metadata. The public validation ladder checks the relevant integer outputs and state words, not only whether $\arg\max z_t$ agrees. Internal comparison points are logical transaction boundaries: an implementation may pipeline or reorder independent work without matching the oracle cycle by cycle, but its accepted transactions and architectural state commits must agree with the oracle.

Listing 1: A state-commit invariant: update architectural state on an accepted transaction. The complete buffered implementation is given in Appendix E.

```
if (out_valid && out_ready)
    state_q <= payload;
```

Three comparisons play different roles. Comparing the framework with an independent FP32 implementation qualifies the mathematical reference under a numerical tolerance; comparing fixed point with FP32 on identical prefixes measures approximation fidelity; and comparing RTL with fixed

point tests the implementation exactly. Fixing an RTL bug cannot improve a fixed-point model's fidelity to FP32. Conversely, imperfect FP32 agreement is not an RTL bug if the RTL faithfully reproduces the integer contract.

3.2 Workloads, assistance and the agent boundary

Table 2 summarizes the Qwen workloads, and Table 1 reports the agent cohorts and paired schedules. Our compact Qwen-derived test model, which we call Qwen3.5-nano (a name of ours, not an official Qwen release), has four layers (three Gated DeltaNet, or GDN, layers and one full-attention layer), hidden width 256, and a 4,096-token vocabulary. The larger tasks adapt the official text configurations (Qwen Team, 2026a;c): Qwen3.5-0.8B has 24 layers and hidden width 1,024, and Qwen3.5-4B has 32 layers and width 2,560, with GDN/full-attention counts of 18/6 and 24/8. The RTL boundary excludes vision, multimodal preprocessing, multi-token prediction, and host sampling. Embedding lookup and the output projection remain within the timed decoder computation.

We distinguish two tracks: *construction* from a supplied interface and *scale adaptation* of a shared, parameterized scaffold. The 0.8B and 4B trajectories are scale adaptation: they establish neither from-scratch synthesis of a billion-parameter architecture nor an agent's independent discovery of the datapath. Within each track, compared agents receive the same starting files, oracle interface, public feedback, and budgets. The harness can invoke lint, compilation, simulation, bounded property checking, and synthesis. Our experiment pipeline uses MathCode (Team Math-AI, 2026) as the agent harness. It records ordered invocations, exit status, elapsed time, file changes, and model usage when available. A run ends on submission or when a frozen budget is exhausted; scale adaptation has a 12-hour wall-clock limit. The hidden evaluation runs only after the agent exits, and timeouts and numerical failures count as failed trials. Appendix G specifies these bindings and distinguishes construction from scaffold adaptation.

3.3 Public ladder, hidden checks and task acceptance

The public ladder proceeds through operators, a GDN layer, a full-attention layer, the full decoder, recurrent decoding, and reset/isolation; larger tasks add 32-step decoding and variable memory latency. First-mismatch feedback reports the layer, token, tensor, word index, and the expected and received integers.

The hidden suite is deliberately modest: different quantized-array seeds and token sequences, 32-step decode, reset between two short sequences, in-order responses with zero to three stall cycles, and parameter reload while idle. These are variations within the contract, not new architectures or adversarial protocol changes. For example, a public reset test can clear a zero state, whereas a hidden test first decodes sequence A, resets, and then checks that sequence B matches a fresh instance. Appendix B specifies the vectors and scope of each check.

Hardware acceptance requires

$$Y = I_{\text{integrity}} \cdot I_{\text{public}} \cdot I_{\text{hidden}} \cdot I_{\text{synth}} \cdot I_{\text{local}}, \tag{2}$$

where I_{local} denotes the local bounded-property checks when a task version requires them. Model fidelity is a workload qualification flag, not a score an agent can improve by modifying a protected oracle. The nano tasks therefore remain useful for hardware debugging even though their fixed-point configuration misses the 99% fidelity requirement. Resource ceilings and timing-screen outcomes are reported separately.

4 Model fidelity, RTL exactness and evaluator checks

Table 2 reports the two comparisons separately. The nano reference moves from 62/64 to 64/64 exact full-model tests and from 27/32 to 32/32 exact eight-token sequences, while its operator and single-layer tests remain fully exact. The remaining numerical failures occur at integration boundaries while the arithmetic definition stays fixed. The final references pass every public and held-out case listed in Table 2. These exact RTL checks coexist with different levels of model fidelity. The nano configuration retains 97.22% pooled top-1 agreement, below the 99% qualification criterion. For 0.8B and 4B, the pooled figures are 99.3125% and 99.2250% over eight positions with 1,000

Table 4: Scale-adaptation trajectories and clean replays have different start states. Agent time includes design iterations from a shared scaffold; blind time begins after the submitted artifact is frozen. Clean replay starts from correct RTL and runs validation and synthesis once each. Invocation counts exclude blind evaluation and include failures.

Workload	Agent h	Blind h	Total h	Calls	Failed	Clean min
Qwen3.5-0.8B	7.25	0.35	7.60	168	11	162.2
Qwen3.5-4B	9.75	0.85	10.60	210	17	225.8

decisions per position, and agreement at the last position is 98.8% in both workloads. All fidelity comparisons use common token prefixes. Twelve targeted defective designs test whether the evaluator rejects common mistakes, including incorrect rounding, truncation, saturation, state ordering, cache position, reset leakage, hard-coded data, fixed latency, unsupported synthesis constructs, and protected-file edits. Public checks reject three of the twelve defects and hidden numerical checks reject ten; together, the gates reject all twelve while accepting the correct control. Appendix B gives the full matrix, and Appendix H visualizes its gate-specific coverage.

5 Agent acceptance under deeper integration

The nano pilot varies integration depth rather than parameter count. T1 constructs a single layer, T2 integrates the four-layer one-step decoder, and T3 adds eight-token stateful decoding. Each task receives five independent attempts from each of three agent-system configurations, for 45 attempts in total. Acceptance falls from 11/15 to 7/15 to 2/15, whereas the corresponding public pass counts are 13/15, 10/15, and 6/15: public success does not imply final acceptance (Table 1, Block A). Each pooled count combines five attempts from each of the three agent configurations and is not a controlled comparison of configurations; within each task, their accepted counts differ by at most one attempt, and with 15 attempts per task the pooled rates are imprecise. Conditioning on public-passing submissions isolates the rejecting gate (Table 1, Block A): all nine rejected submissions fail hidden numerical checks while passing synthesis and integrity. Their recorded diagnoses comprise two arithmetic, three cross-layer integration, and four recurrent-state failures; in T3, three of the four rejections carry a state diagnosis.

The larger-model case studies answer a different question: does the scaffold support a complete accepted trajectory? The 0.8B and 4B agents exit after 7.25 and 9.75 hours, and the blind evaluator finishes at 7.60 and 10.60 hours; Table 4 also lists their tool calls, and Appendix D gives the 0.8B milestone trace (Listing 3) and the invocation schema.

A clean replay has a different denominator: it starts from correct RTL and runs validation and synthesis once, taking 162.2 minutes for 0.8B and 225.8 minutes for 4B, of which synthesis and timing take 31 and 44 minutes. Empirical cost per accepted run sums the costs of *all* attempts, including failures, and divides by the number of accepted attempts; Appendix C reports this quantity rather than median cost divided by a success fraction.

Breadth across decoder contracts. Additional reference checks cover Gemma 2 2B, the Gemma 3 4B text decoder, Qwen3.6-27B, and Llama 3.2-1B (Table 12). The Gemma and Llama references satisfy their finite integer suites, although Gemma 2 2B remains below the 99% fidelity criterion. Qwen3.6-27B passes its operator and layer checks but still fails full-model and recurrent cases, so operator-level compatibility does not establish complete integration at a larger configuration. These checks broaden the tested target semantics without adding agent attempts; Appendix J identifies each model and its test allocation.

Coding-agent comparison and state-feedback ablation. We compare Sol 5.6 and Astra 6 within MathCode, holding the task instances, starting files, instructions, tools, feedback policy, and stopping rules fixed within each task. The comparison contains 16 fresh attempts per agent at each nano integration level and 12 scale-adaptation attempts per agent on Qwen3.5-0.8B (Table 5, Block A).

The comparison is mixed. Sol accepts one more single-layer attempt, whereas Astra accepts one more full-model attempt and two more stateful attempts; both agents accept fewer attempts as temporal obligations increase. On 0.8B, the agents accept 7/12 and 8/12 submissions, respectively. Each

Table 5: Sol 5.6 and Astra 6 under MathCode. The blocks answer different questions: A compares agents, and B varies public feedback with the agent fixed. **A.** Independent attempts with output-and-state public feedback; the two agents share the task instances, starting files, instructions, tools, and per-task limits. Counts are out of all attempts in each cell (16 per nano task, 12 for Qwen3.5-0.8B); *public* means the canonical state-aware public suite passed and *final* means every final gate passed. Hours per accepted submission divide total agent wall time, including failed and timed-out attempts, by the number accepted. **B.** Sol 5.6 with two public-feedback policies on the same 16 task instances per task (policies defined in Appendix L). *Visible* follows the feedback policy, while *canonical* and *final* are computed after submission with every state check. The output-and-state rows reuse the Sol runs of Block A. Blue shading encodes n/N on one scale.

A. Agent comparison (output-and-state public feedback)

	Nano T1 single layer		**Nano T2** full model		**Nano T3** stateful decode		**Qwen3.5-0.8B** scale adaptation	
Agent	Public	Final	Public	Final	Public	Final	Public	Final
Sol 5.6	14/16	13/16	12/16	10/16	9/16	5/16	9/12	7/12
Astra 6	14/16	12/16	13/16	11/16	10/16	7/16	10/12	8/12
Sol 5.6 h/accepted	2.32		4.76		15.14		14.53	
Astra 6 h/accepted	2.48		4.91		10.09		12.20	

B. State-feedback ablation (Sol 5.6 fixed; same evaluator after submission)

	Nano T2 full model			**Nano T3** stateful decode		
Public feedback	Visible	Canonical	Final	Visible	Canonical	Final
Output only	13/16	10/16	8/16	10/16	7/16	3/16
Output and state	12/16	12/16	10/16	9/16	9/16	5/16

paired contrast rests on three to five discordant task instances (Appendix L), so these small cohorts support a comparison of failure patterns and costs rather than a stable ranking of the coding models.

To examine the contribution of state observability, we keep Sol fixed and compare output-only public feedback with output-and-state feedback; the final evaluator retains every state check in both conditions. State feedback raises final acceptance from 8/16 to 10/16 on T2 and from 3/16 to 5/16 on T3 (Table 5, Block B). Under output-only feedback, 10/16 T3 submissions satisfy the visible checks, but only 7/16 pass the canonical state-aware public recheck and 3/16 pass final acceptance.

Construction cost counts failed attempts. On T3, total agent time per accepted submission is 15.14 hours for Sol and 10.09 hours for Astra under state feedback, and 26.82 hours for Sol under output-only feedback. The accepted 0.8B implementations of the two agents occupy overlapping throughput and LUT ranges (Appendix L).

6 Hardware Efficiency under Fixed Arithmetic

6.1 Scheduling Improves Throughput Without Expanding the Datapath

All three designs time-multiplex a shared compute engine and stream weights through an external-memory interface. Scaling the model therefore increases the number of executed operations, the amount of persistent state, and memory traffic, but leaves the instantiated arithmetic datapath largely unchanged. To isolate the effect of scheduling, we compare each baseline and optimized design under the same fixed-point contract, 126 MHz target clock, batch size of one, eight-token prompt, and memory-service model. Throughput is $R = f/C$, where f is the target clock frequency and C counts all elapsed cycles per decoded token, including memory stalls.

In all three workloads, the optimized schedules reduce cycles per token without materially increasing mapped logic. Throughput improves by 15.0% for both Qwen3.5-nano and Qwen3.5-0.8B and by

12.5% for Qwen3.5-4B, while LUT counts change by less than 0.2%. We therefore read the result as a scheduling improvement at approximately fixed logic cost, not as evidence of area reduction. Other resources are not exactly constant: some configurations use slightly more registers and on-chip memory to buffer data and overlap transfers with computation.

The 4B cycle breakdown in Figure 3b shows where elapsed cycles decrease. External-memory waiting falls from 47.0 million to 34.5 million cycles per token, whereas matrix computation changes only modestly, from 86.0 million to 83.0 million cycles. This profile is consistent with the optimized schedule hiding a larger fraction of memory latency, rather than gaining from additional arithmetic parallelism. The counters assign every cycle to exactly one owner under a fixed attribution rule, so overlapping computation and transfer are not double-counted.

The shared, time-multiplexed architecture also explains why logic footprints are similar across model sizes. The optimized 0.8B and 4B instances both use 128 DSP blocks, while their BRAM36 use increases from 224 to 248 blocks as the state and buffering requirements grow. These buffers are far too small to store the full model weights, so both designs depend on external storage.

6.2 From throughput to energy efficiency

Fewer cycles per token do not guarantee lower energy: a faster schedule may increase device or memory power. We therefore specify the system boundary explicitly. Useful work is batch-one decode output, excluding prompt processing, and average power P includes both the FPGA and its attached DRAM; host execution, network communication, and the model-generation API lie outside this boundary. For decode throughput R, we report $E_{\text{token}} = \frac{P}{R}, \eta = \frac{R}{P}$ tokens/J, $\eta_{\text{kW}} = \frac{1000R}{P}$ tokens/s/kW. Including the attached memory matters here because all three accelerators stream weights from external storage; device-only power would omit part of the cost that the architecture itself creates. The energy ratio $\frac{E^{\text{opt}}_{\text{token}}}{E^{\text{base}}_{\text{token}}} = \frac{P^{\text{opt}}/P^{\text{base}}}{R^{\text{opt}}/R^{\text{base}}}$ then separates the contributions of speed and power. With the power inputs in Block B of Table 1, the optimized schedules improve energy efficiency by 7.1–11.3% across the three workloads, and energy per token falls from 0.0164 to 0.0148 J (Qwen3.5-nano), from 2.2500 to 2.0435 J (Qwen3.5-0.8B), and from 12.5000 to 11.6667 J (Qwen3.5-4B). These gains are smaller than the corresponding throughput improvements because the optimized designs draw modestly more average boundary power.

Our analysis characterizes inference within the stated FPGA-plus-DRAM boundary. Comparisons with commercial accelerators would likely also require matched model quality, precision, sequence lengths, batch size, memory traffic, and measurement boundaries, so we restrict the present comparison to paired implementations of the same workload. Appendix F describes the power methodology, and Appendix K gives per-workload efficiency plots and power-sensitivity curves.

7 Conclusion

We introduced AIChipBench, a state-aware benchmark for coding agents that translate AI model specifications into synthesizable RTL. Its central premise is that AI acceleration should be treated as implementing an executable state-transition system, so correctness covers not only the current token but also the recurrent, convolutional, and attention states that determine later outputs. Checking these transitions against a frozen integer oracle at progressively deeper levels of integration exposes errors that compilation, synthesis, or next-token agreement alone can miss, and it keeps model fidelity, RTL exactness, and agent acceptance beyond public feedback as separate questions.

Our Qwen reference designs address feasibility, while acceptance in the nano cohort decreases as cross-layer and temporal obligations grow. Gate outcomes place the public-to-final gap in the hidden numerical checks; the reduced replay demonstrates one mechanism for delayed observability without attributing every rejection to it. Because such failures can surface several layers or transactions after their cause, the benchmark can also be used to study how agents localize faults, use simulator feedback, preserve invariants, and allocate tool budget. Once numerical behavior is fixed, scheduling improves decode throughput without materially expanding the arithmetic datapath, but the resulting efficiency claim depends on the surrounding memory system and its power cost. The current tasks cover a controlled family of text decoders, formal checks are local and bounded, and the larger workloads adapt a shared parameterized scaffold.

AI USE STATEMENT

AI assistance was used for manuscript drafting, polishing writing, and reference discovery. It was also used in figure preparation: an image-generation tool produced the initial designs of earlier versions of the overview and workflow illustrations, AI assistance was used to implement those designs as TikZ figures, an AI coding assistant drew the current Figures 1 and 4 in TikZ from the same icon set, and an AI coding assistant restyled the typography of the result plots and generated several result tables and plots from the repository records. The authors retain responsibility for all claims, code, citations, and compliance with the venue's policies.

ETHICS STATEMENT

The benchmark does not evaluate human subjects or require private training data. Shared traces omit credentials, proprietary tool content, and private reasoning, retaining only observable actions and outputs permitted by their licenses. We ensure that release procedures preserve the separation between agent-visible assets and held-out tests.

REPRODUCIBILITY STATEMENT

Section 3 specifies the workload contracts, numerical references, and acceptance procedure. Appendices A and G describe the configuration requirements, starting artifacts, and assistance available to each agent. Appendix B details the public and held-out checks, including their supported input, state, and memory-interface variations. Success fractions and construction costs are defined in Appendix C, with the execution-trace schema in Appendix D and representative RTL implementations in Appendix E. Hardware comparisons hold arithmetic and the memory-service model fixed within each workload and use a 126 MHz target clock, batch size one, and an eight-token prompt. Section 6.2 specifies the energy boundary, Block B of Table 1 lists the power inputs used in the efficiency analysis, and Appendix F describes the power methodology. These specifications distinguish numerical fidelity, RTL correctness, agent construction reliability, and hardware quality so that each outcome can be reproduced and assessed separately.

REFERENCES

Ahmed Allam and Mohamed Shalan. RTL-Repo: A benchmark for evaluating LLMs on large-scale RTL design projects. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pp. 1–5, 2024. doi: 10.1109/LAD62341.2024.10691810.

AMD. Vivado design suite user guide: Power analysis and optimization (UG907), 2026. URL https://docs.amd.com/r/en-US/ug907-vivado-power-analysis-optimization. Switching activity and implementation-stage power analysis; accessed September 17, 2026.

Architect Labs. Functional RTL generation: A study in curriculum learning and reward shaping with limited data. Technical blog, May 2026. URL https://architectlabs.com/blog/functional-rtl-generation.

atopile. EEBench: Methodology, 2026. URL https://eebench.org/methodology.html. Accessed September 22, 2026.

Lei Chen, Yiqi Chen, Zhufei Chu, Wenji Fang, Tsung-Yi Ho, et al. Large circuit models: Opportunities and challenges. *Science China Information Sciences*, 67(10):200402, 2024. doi: 10.1007/s11432-024-4155-7. Preprint: The Dawn of AI-Native EDA: Opportunities and Challenges of Large Circuit Models, arXiv:2403.07257.

Dario Garcia-Gasulla, Gokcen Kestor, Emanuele Parisi, Miquel Albertí-Binimelis, Cristian Gutierrez, Razine Moundir Ghorab, Orlando Montenegro, Bernat Homs, and Miquel Moreto. TuRTLe: A unified evaluation of LLMs for RTL generation. In *2025 ACM/IEEE 7th Symposium on Machine Learning for CAD (MLCAD)*, pp. 1–12, 2025. doi: 10.1109/MLCAD65511.2025.11189228.

Google. Gemma 2 model card, 2024. URL https://ai.google.dev/gemma/docs/core/model_card_2.

Google DeepMind. Gemma 3 model card, 2025. URL https://ai.google.dev/gemma/docs/core/model_card_3.

LBC-bench Team. LBC-bench leaderboards, 2025. URL https://lbc-bench.si2.org. Hosted at lbc-bench.si2.org; website source: https://github.com/si2org/lbc-bench.github.io; accessed September 25, 2026.

Mingjie Liu, Teodor-Dumitru Ene, Robert Kirby, Chris Cheng, Nathaniel Pinckney, et al. ChipNeMo: Domain-adapted LLMs for chip design. arXiv:2311.00176, 2023a. URL https://arxiv.org/abs/2311.00176.

Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. VerilogEval: Evaluating large language models for Verilog code generation. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–8. IEEE, 2023b. doi: 10.1109/ICCAD57390.2023.10323812. Invited paper.

Shang Liu, Yao Lu, Wenji Fang, Mengming Li, and Zhiyao Xie. OpenLLM-RTL: Open dataset and benchmark for LLM-aided design RTL generation. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–9. ACM, 2024. doi: 10.1145/3676536.3697118. URL https://arxiv.org/abs/2503.15112. Invited paper.

Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. RTLLM: An open-source benchmark for design RTL generation with large language model. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 722–727. IEEE, 2024. doi: 10.1109/ASP-DAC58780.2024.10473904. URL https://arxiv.org/abs/2308.05345.

Meta. Llama 3.2 model card, 2024. URL https://huggingface.co/meta-llama/Llama-3.2-1B.

Nathaniel Pinckney, Chenhui Deng, Chia-Tung Ho, Yun-Da Tsai, Mingjie Liu, Wenfei Zhou, Brucek Khailany, and Haoxing Ren. Comprehensive Verilog design problems: A next-generation benchmark dataset for evaluating large language models and agents on RTL design and verification. arXiv:2506.14074, 2025. URL https://arxiv.org/abs/2506.14074.

Qwen Team. Qwen3.5-0.8B: Model card and configuration, 2026a. URL https://huggingface.co/Qwen/Qwen3.5-0.8B. Hugging Face model repository (model card and config.json); accessed September 26, 2026.

Qwen Team. Qwen3.6-27B: Model card and configuration, 2026b. URL https://huggingface.co/Qwen/Qwen3.6-27B. Hugging Face model repository (model card and config.json); accessed September 26, 2026.

Qwen Team. Qwen3.5-4B: Model card and configuration, 2026c. URL https://huggingface.co/Qwen/Qwen3.5-4B. Hugging Face model repository (model card and config.json); accessed September 26, 2026.

Team Math-AI. MathCode: A frontier mathematical coding agent. math-ai-org.github.io, April 2026. URL https://github.com/math-ai-org/mathcode.

Henry Wong. HDLBits: Verilog practice, 2024. URL https://hdlbits.01xz.net/wiki/Main_Page. Accessed September 17, 2026.

Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. ChatEDA: A large language model powered autonomous agent for EDA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(10):3184–3197, 2024. doi: 10.1109/TCAD.2024.3383347. URL https://arxiv.org/abs/2308.10204.

Zhongkai Yu, Chenyang Zhou, Yichen Lin, Hejia Zhang, Haotian Ye, Junxia Cui, Zaifeng Pan, Jishen Zhao, and Yufei Ding. ChipBench: A next-step benchmark for evaluating LLM performance in AI-aided chip design. arXiv:2601.21448, 2026. URL https://arxiv.org/abs/2601.21448.

Brosnan Yuen and Minh Tu Hoang. EDASYTH: An LLM-in-the-loop hardware design workflow with LTspice and KiCad. Preprints.org preprint, version 1, posted September 11, 2026, doi:10.20944/preprints202609.0945.v1; not peer reviewed, 2026. URL https://www.preprints.org/manuscript/202609.0945.

A EXPERIMENTS AND BENCHMARK BOUNDARIES

The benchmark has two independently versioned objects: a workload contract and an agent-run configuration. The run-manifest fields in Table 6 determine which comparison a reported result supports.

Table 6: Required run-manifest fields and their roles.

Field	Meaning and purpose
Task revision	Specification, public tests, hidden-test policy and RTL interface version
Model revision	Exact checkpoint and configuration; tokenizer and export software versions
Arithmetic digest	Bit widths, scales, tile order, nonlinear tables, rounding and saturation
Starting artifact	Blank-start or scaffold-adaptation track; files available at time zero
Agent configuration	Provider model identifier, sampling settings, prompt, harness revision
Budgets	Wall time, generated tokens, tool/evaluator calls and timeout policy
Execution environment	CPU, RAM, operating system, simulator, compiler, synthesis version
Hardware constraints	Device/part, clocks, memory interface, resource caps, I/O exclusions
Isolation	Hidden assets unavailable during generation; protected files read-only
Output identity	Hashes of submitted RTL, reports and artifacts after agent exit
Outcome	Each gate separately, first mismatch, cost and termination reason

What is held fixed. The paired schedule comparison within a workload uses the same integer functions, parameter arrays, sequence lengths, external-memory behavior, and 126 MHz conversion. The RTL may change buffering, overlap, and scheduling, but not approximations or matrix dimensions. A change in precision, a new nonlinear table, or a different accumulation order requires a new arithmetic version, even if next-token agreement happens to improve.

Supported scope. The nano configuration has four layers, hidden width 256, intermediate width 768 and vocabulary 4,096. For 0.8B the corresponding dimensions are 24, 1,024, 3,584 and 248,320; for 4B they are 32, 2,560, 9,216 and 248,320. The larger-model text configurations include attention output gating, convolution length four and a full-attention layer every fourth layer.

Reference independence. The independent FP32 implementation is compared with the framework at operator, layer, full-forward, and recurrent boundaries, using identical weights and input prefixes. Tests cover state tensors, not just final logits. The integer oracle uses explicit integer arithmetic for every fixed-point stage, and its rounding and saturation are checked against a second implementation, such as exact rational arithmetic.

B CONCRETE PUBLIC AND HIDDEN TESTS

The public suite teaches the interface and provides reproducible feedback. The hidden suite changes seeds and short sequences without introducing unsupported operations. A test case is a complete stimulus/check pair; the tensor elements within it are not independent statistical trials. One mismatch fails a case, and one failed required case fails the numerical task gate. "Long decode" here means 32 steps, not thousands of tokens.

Weight and input generalization. "Unseen weights" means new quantized arrays generated within the supported shapes and ranges. Such tests are useful for catching hard-coded vectors and stale weight buffers. Numerical fidelity to FP32 is evaluated separately, on checkpoint-derived weights. Randomizing all scales arbitrarily would violate the frozen arithmetic contract and is not part of the test distribution.

Fidelity aggregation. At each of eight positions, the 0.8B table counts 997, 996, 995, 994, 993, 992, 990 and 988 matching top-1 decisions out of 1,000. This totals 7,945/8,000. The corresponding 4B counts are 996, 995, 994, 993, 992, 991, 989, and 988, totaling 7,938/8,000. Nano uses 256 sequences per position and totals 1,991/2,048. These comparisons use matched token histories, while the FP32 and integer states each evolve under their own arithmetic.

Table 7: Public and hidden tests share their documented semantics.

Check	Public tests	Held-out variation and expected behavior
Rounding	Shift by 8: $128 \mapsto 0, 384 \mapsto 2$	Negative ties: $-128 \mapsto 0$, $-384 \mapsto -2$; ties to even
Saturation	Largest representable INT16 result	Positive and negative overflow clamp to $32{,}767$ and $-32,768$
Token sequence	Prefill IDs $[1, 7, 13]$; decode eight steps	Different in-range IDs and prompt length five; all logits/state exact
Weight reload	Load set A into an idle engine	Load shape-compatible quantized set B; no stale A values
Recurrent state	Zero initial GDN and convolution state	Small supported nonzero state; check state after every token
Reset/isolation	Reset before first sequence	Decode A, reset, decode B; compare B with a fresh instance
Backpressure	Always-ready in-order responses	Stall for 1–3 cycles; stable payload, no loss or duplicate commit
Cache address	Start from position zero	Nonzero supported position and a short prefill/decode transition
Integrity	Read-only oracle and constraints	Reject modifications; run evaluator from an independent clean copy

Table 8: Evaluator sanity checks on twelve targeted defects and one correct control. True means a gate accepts that design.

Defect	Public	Hidden	Synthesis	Integrity
Correct reference	True	True	True	True
Wrong rounding mode	False	False	True	True
Signed accumulator truncation	False	False	True	True
Saturation off by one	True	False	True	True
GDN update before decay	True	False	True	True
Conv-state late writeback	False	False	True	True
KV-cache position offset	True	False	True	True
Reset leakage	True	False	True	True
Public-vector hard-code	True	False	True	False
Hidden group-size hard-code	True	False	True	True
Fixed memory-latency assumption	True	False	True	True
Non-synthesizable lookup	True	True	False	True
Protected evaluator edit	True	True	True	False

Local formal scope. The controller checks assume a single clock, reset at the initial step, in-order memory responses, and response stalls of at most three cycles. Safety is checked for 32 cycles on reduced control instances: output payload is stable while blocked; one accepted request commits at most once; reset clears valid state; issued addresses are in the configured range; and reload occurs only while idle. Unbounded liveness, full-network equivalence, CDC/RDC, and arbitrary response reordering are outside these checks.

Evaluator sanity checks. Table 8 gives the complete acceptance matrix. "True" means that the gate accepts the design, even when the design is defective. A correct control must pass every gate, and each mutant must be rejected by at least one gate. Detecting these twelve defects shows that the checks are wired into acceptance, not that the hidden tests cannot be overfit. Protected-file hashing and test isolation address different risks, and both are necessary.

C Cost, success fractions and runtime accounting

For a fixed task and agent-system configuration, independent runs yield binary outcomes Y_i, and the empirical fraction is $\hat{p} = \sum_i Y_i/N$. A single success $(1/1)$ is still an observation, but its uncertainty is too large to support a reliability claim. The nano pilot has five attempts per system per task:

Table 9: Nano-task hardware acceptance and empirical cost accounting. Total cost per accepted run includes all failed and timed-out runs in the numerator.

Task	Pass	Runs	Median h/run	Total h/pass	Tokens/pass
T1 Single layer	11	15	1.76	2.59	196,053
T2 Full model	7	15	3.07	7.43	546,833
T3 Stateful decode	2	15	3.58	30.23	2,839,580

Codex-system counts are 4/5, 3/5, 1/5; Claude-system counts are 4/5, 2/5, 1/5; Kimi-system counts are 3/5, 2/5, 0/5. Provider names here identify configurations, not a controlled comparison of base model weights.

For observed costs c_i, define

$$\widehat{c}_{\text{accepted}} = \frac{\sum_{i=1}^{N} c_i}{\sum_{i=1}^{N} Y_i}. \tag{3}$$

This ratio retains the costs of failed attempts; if no attempt passes, it is undefined rather than zero. It is an empirical accounting quantity, and interpreting it as the expected cost of future retries requires an appropriate repeat-run distribution. Dividing median cost by $\hat{p}$ generally gives a different value. Table 9 uses the full run-level costs.

D TRACE FORMAT AND A STATE-LOCALIZED REPAIR

The event schema is line-delimited JSON with a monotonically increasing event identifier, elapsed time, actor, event type, and payload. Tool start and end events share a call identifier and a tool category. A failed invocation counts in its category and in the failure subtotal, not in a separate, mutually exclusive category. High-level action summaries suffice for audit; private model reasoning is neither required nor reconstructed. Token usage is recorded only from provider-reported usage; unavailable values are left absent rather than estimated from wall time.

Listing 2: Observable trace schema at the submission boundary. Elapsed time is in seconds. The final evaluator is a separate actor, and its response is not fed back to the generating agent.

```
{"event_type":"agent_exit","actor":"agent",
 "elapsed_s":26100,
 "payload":{"tool_calls":168,"failed_calls":11,
            "public_exact":3120,"public_total":3120}}
{"event_type":"hidden_evaluation_start","actor":"grader",
 "elapsed_s":26100,"payload":{"feedback_to_agent":false}}
{"event_type":"hidden_evaluation_end","actor":"grader",
 "elapsed_s":27360,
 "payload":{"numerical_exact":232,"numerical_total":232,
            "integrity":true}}
```

Listing 3 summarizes the same 0.8B run as cumulative milestones, keeping stage completions and the agent/grader boundary.

Listing 3: 0.8B scale-adaptation milestones. Times are cumulative wall-hours.

```
3.70  compile milestone: first compilable RTL
5.40  public validation ladder: 3120/3120 exact
6.20  schedule optimization complete
6.70  local bounded checks complete
7.25  synthesis complete; agent exits; artifact frozen
7.60  grader: hidden numerical 232/232; integrity pass
```

The 0.8B tool counts are 42 reads, 27 edits, 50 shell/Python calls, 8 lint calls, 12 compilations, 19 simulations, 6 formal invocations, and 4 synthesis invocations, with failure subtotals of 0, 0, 2, 1, 3, 3, 1, and 1. The 4B totals are 50, 33, 61, 11, 15, 27, 9, and 4, with 0, 0, 3, 2, 4, 5, 2, and 1 failures. These sum to 168/11 and 210/17 total/failed invocations. The final hidden evaluation is outside these agent-side counts. File edits and RTL-changing iterations need not be equal, because one iteration can modify several files.

Listing 4: State commits are tied to accepted transactions. The payload remains stable while valid is asserted and ready is low; the complete elastic-buffer implementation also handles simultaneous replacement and reset.

```
if (out_valid && out_ready)
    state_q <= payload;
```

A useful first-mismatch record. The public evaluator returns a record reporting layer 3, token 2, convolution-state word 7, expected integer 127, and received integer 126 at the accepted state-write transaction. The record shows where debugging starts, not the proven cause; candidate explanations include rounding, stale data, or an incorrect write enable. A repair must be rechecked first on a small reproducer, then on the affected layers, and finally on the complete frozen suite. A trace that merely reports "state bug fixed" loses the evidence needed to distinguish these explanations.

The complete component in Appendix E ties state updates to accepted transactions. Updating state whenever a result is valid, without waiting for the transaction to be accepted, can commit the same value repeatedly or desynchronize a downstream state machine under backpressure.

E Arithmetic and transactional RTL components

The following modules are complete at their stated interfaces.

Nearest-even requantization. For shift k, write $x = 2^k q + r$ with $0 \le r < 2^k$ and $q = \lfloor x/2^k \rfloor$. Increment q when $r > 2^{k-1}$, or on an exact tie when q is odd; then saturate. This formulation also handles negative ties, because an arithmetic right shift computes the floor quotient. Taking the absolute value first would overflow at the most negative representable input and would need special handling. The module's legal domain is $1 \le \texttt{SHIFT} < \texttt{IN_W}$ and $2 \le \texttt{OUT_W} \le \texttt{IN_W}$.

Listing 5: Complete signed nearest-even requantizer with output saturation. The one-bit-wider intermediate preserves the rounding increment before comparison with the output limits.

```
// Signed power-of-two requantization: round to nearest, ties to even, saturate.
// Legal parameters: 1 <= SHIFT < IN_W, 2 <= OUT_W <= IN_W.
module requant_rne_sat #(
    parameter integer IN_W = 32,
    parameter integer OUT_W = 16,
    parameter integer SHIFT = 8
) (
    input  wire signed [IN_W-1:0] x,
    output reg  signed [OUT_W-1:0] y
);
    localparam signed [IN_W:0] MAX_Y =
        ({{IN_W{1'b0}},1'b1} << (OUT_W-1)) - 1'b1;
    localparam signed [IN_W:0] MIN_Y =
        -({{IN_W{1'b0}},1'b1} << (OUT_W-1));
    localparam [SHIFT-1:0] HALF = {1'b1,{(SHIFT-1){1'b0}}};
    reg signed [IN_W:0] floor_q;
    reg signed [IN_W:0] rounded_q;
    reg [SHIFT-1:0] remainder;
    reg increment;
    always @* begin
        // Arithmetic shift is floor division, also for negative inputs.
        floor_q = $signed({x[IN_W-1],x}) >>> SHIFT;
        remainder = x[SHIFT-1:0];
        increment = (remainder > HALF) ||
                    ((remainder == HALF) && floor_q[0]);
        rounded_q = floor_q + {{IN_W{1'b0}},increment};
        if (rounded_q > MAX_Y)
            y = {1'b0,{(OUT_W-1){1'b1}}};
        else if (rounded_q < MIN_Y)
            y = {1'b1,{(OUT_W-1){1'b0}}};
        else
            y = rounded_q[OUT_W-1:0];
    end
endmodule
```

State commit and cancellation. The one-entry buffer accepts a new pending state word when it is empty or when the current word is consumed, and architectural state changes on output acceptance. A simultaneous pop/push commits the old payload and retains the new one, using nonblocking assignments. Reset or a new-sequence request cancels pending work and clears the state. A system with a different reset policy must change the contract and the tests together.

Listing 6: Complete one-entry state-commit buffer. Inputs and outputs use ready/valid handshakes on one clock.

```
// One-entry elastic buffer for a completed architectural state word.
// State changes only when the downstream transaction is accepted.
// Reset/new_sequence cancels pending work and clears architectural state.
module state_commit_buffer #(
    parameter integer W = 16
) (
    input wire clk,
    input wire rst,
    input wire new_sequence,
    input wire in_valid,
    output wire in_ready,
    input wire [W-1:0] in_data,
    output wire out_valid,
    input wire out_ready,
    output wire [W-1:0] out_data,
    output reg [W-1:0] state_q
);
    reg pending;
    reg [W-1:0] payload;
    wire cancel = rst || new_sequence;
    assign in_ready = !cancel && (!pending || out_ready);
    assign out_valid = !cancel && pending;
    assign out_data = payload;
    always @(posedge clk) begin
        if (cancel) begin
            pending <= 1'b0;
            payload <= {W{1'b0}};
            state_q <= {W{1'b0}};
        end else begin
            if (out_valid && out_ready) begin
                state_q <= payload;
                pending <= 1'b0;
            end
            // A new item can replace the just-committed item in one cycle.
            // Nonblocking assignments commit the old payload above.
            if (in_valid && in_ready) begin
                payload <= in_data;
                pending <= 1'b1;
            end
        end
    end
endmodule
```

Listing 7: Directed component testbench. It covers positive and negative rounding ties, saturation, enqueue without commit, a three-cycle stall, simultaneous pop/push, and reset isolation.

```
`timescale 1ns/1ps
module tb_components;
    reg clk=0;
    always #5 clk=~clk;
    reg signed [31:0] x;
    wire signed [15:0] y;
    requant_rne_sat dut_q(.x(x),.y(y));
    reg rst=1, new_sequence=0, in_valid=0, out_ready=0;
    reg [15:0] in_data=0;
    wire in_ready, out_valid;
    wire [15:0] out_data,state_q;
    state_commit_buffer dut_s(.*);
    task qcheck(input integer value, input integer expected);
        begin
            x=value; #1;
            if ($signed(y) !== expected)
                $fatal(1,"requant mismatch x=%0d got=%0d expected=%0d",value,$signed(y),
     expected);
        end
    endtask
    task tick;
        begin @(posedge clk); #1; end
```

```
    endtask
    initial begin
        qcheck(128,0); qcheck(384,2); qcheck(-128,0);
        qcheck(-384,-2); qcheck(32767*256+128,32767);
        qcheck(-32768*256-256,-32768);
        qcheck(2147483647,32767); qcheck(-2147483647-1,-32768);
        @(negedge clk); rst=0; in_valid=1; in_data=16'h1234;
        tick();
        if (!out_valid || out_data!==16'h1234 || state_q!==0)
            $fatal(1,"enqueue must not commit");
        @(negedge clk); in_valid=0;
        repeat (3) begin
            tick();
            if (!out_valid || out_data!==16'h1234 || state_q!==0)
                $fatal(1,"stalled output changed");
        end
        @(negedge clk); out_ready=1; in_valid=1; in_data=16'h5678;
        tick();
        if (state_q!==16'h1234 || out_data!==16'h5678 || !out_valid)
            $fatal(1,"simultaneous pop/push wrong");
        @(negedge clk); in_valid=0;
        tick();
        if (state_q!==16'h5678 || out_valid) $fatal(1,"second commit wrong");
        @(negedge clk); new_sequence=1;
        tick();
        if (state_q!==0 || out_valid || in_ready) $fatal(1,"reset leakage");
        $display("PASS: requantization and state-commit directed checks");
        $finish;
    end
endmodule
```

F Power methodology and timing interpretation

Three non-interchangeable evidence levels. A power budget links power inputs and work counts to energy. A post-synthesis power report estimates activity-dependent power under explicit clock, voltage, temperature, process, and switching assumptions, and a post-route report additionally uses physical implementation information. Vendor guidance emphasizes both switching annotation and implementation information (AMD, 2026).

Sensitivity. Let $g_R = R_1/R_0$ and $g_P = P_1/P_0$. The energy-efficiency gain is $g_R/g_P - 1$. Consequently, a 15% throughput gain yields no energy benefit if power also rises by 15%, and the 12.5% throughput gain breaks even at a 12.5% power increase.

G Task configuration and execution records

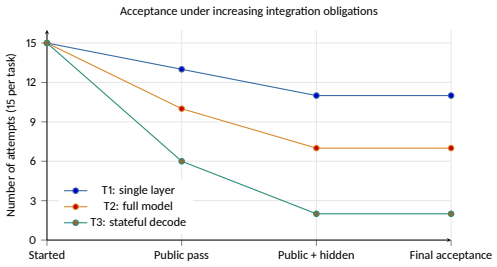
Starting files and assistance. The initial workspace determines how much implementation work the agent must perform and is therefore part of the task definition. For construction tasks, this includes the supplied module declarations, testbench interfaces, arithmetic components, and any working RTL. For adaptation tasks, it also includes the initial accelerator tree and its supported configuration. To make these assistance conditions reproducible, the protocol requires a pre-run inventory of all readable and editable paths, including any accessible reference implementations. Separate digests identify the public tests, oracle code, arithmetic tables, held-out evaluator assets, and hardware constraints. Comparisons within a task hold the starting implementation fixed across agents. Changing the supplied implementation defines a different assistance condition, even when the textual instruction remains unchanged.

Agent configuration and run accounting. An agent configuration includes both the model and the execution settings that govern its behavior. The run specification therefore requires the provider-returned model identifier, its revision when available, the complete system and user prompts, the harness revision, reasoning and sampling settings, the context-management policy, and stopping criteria. Wall-time, token, and tool-call limits are specified separately so that each resource constraint is explicit. Termination labels distinguish API errors and infrastructure interruptions from RTL failures. Run accounting is maintained separately for the nano construction cohort and the larger scaffold-adaptation cases, reflecting their different starting conditions. Each cohort's ledger retains

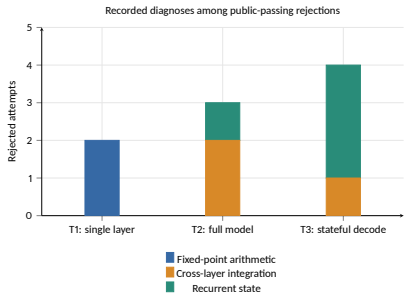
all initiated attempts, including failures and timeouts. A continuation remains part of its original attempt, so resuming a run does not create an additional independent trial.

H CONDITIONAL ACCEPTANCE AND EVALUATOR DIAGNOSIS

The public-to-final difference is a count of submissions, not a count of failing tests. Figure 5a follows the same submissions through the gates within each task, while Figure 5b partitions only the public-passing rejections. The public-pass rejection counts in Block A of Table 1 use the same public-passing denominator. A submission can fail more than one gate, so individual gate failures are retained even when a reporting rule selects one primary rejection. For this nano cohort, the local-property factor is not an additional gate: the 36/44 local checks belong to the larger adaptation cases and cannot be retroactively attached to the nano attempts.



(a) Acceptance of the same submissions through the evaluation gates. Each task starts with 15 fresh attempts. Public passing counts are 13, 10 and 6, and final accepted counts are 11, 7 and 2. The flat final segments indicate no additional synthesis or integrity rejection among the hidden-passing submissions in this cohort; task levels are independent runs.



(b) Recorded diagnoses among the nine public-passing rejections. Counts partition the primary labels: two arithmetic, three cross-layer integration and four recurrent-state failures. Only three of the four T3 rejections have a recurrent-state label. These diagnoses do not establish that every public-to-final failure is caused by persistent state.

	Public	Hidden	Synthesis	Integrity	Final
Correct reference	Pass	Pass	Pass	Pass	Pass
Wrong rounding mode	Reject	Reject	Pass	Pass	Reject
Signed accumulator truncation	Reject	Reject	Pass	Pass	Reject
Saturation off by one	Pass	Reject	Pass	Pass	Reject
GDN update before decay	Pass	Reject	Pass	Pass	Reject
Conv-state late writeback	Reject	Reject	Pass	Pass	Reject
KV-cache position offset	Pass	Reject	Pass	Pass	Reject
Reset leakage	Pass	Reject	Pass	Pass	Reject
Public-vector hard-code	Pass	Reject	Pass	Reject	Reject
Hidden group-size hard-code	Pass	Reject	Pass	Pass	Reject
Fixed memory-latency assumption	Pass	Reject	Pass	Pass	Reject
Non-synthesizable lookup	Pass	Pass	Reject	Pass	Reject
Protected evaluator edit	Pass	Pass	Pass	Reject	Reject

(c) Acceptance matrix for a control and twelve targeted defects. Pass means a gate accepts the design, including a defective one; any required rejection prevents final acceptance. The gate outcomes overlap, so rejection counts cannot be summed across columns. The matrix tests these explicit defects rather than claiming coverage over arbitrary implementations.

Figure 5: Public-to-final gate outcomes and targeted-defect coverage.

Table 10 retains the identities of all nine public-passing rejections and their recorded diagnoses.

Mutation coverage has its own denominator. The mutation set comprises twelve specified defective implementations plus a correct control; it is not a random sample of all RTL bugs. Public checks reject three mutants and hidden checks reject ten, and combining the numerical, synthesis, and integrity gates rejects all twelve while accepting the control (Figure 5c). The checks are thus complementary on these defects, but this does not estimate detection sensitivity in general. The timing of first detection must be tied to an activated defect and an identified stimulus, and a mutation that is never activated within the test horizon is reported separately from one that executes yet escapes all assertions.

Table 10: All public-passing submissions that fail final acceptance in the nano cohort. Each row fails hidden numerical checks while passing synthesis and integrity. Failure labels are recorded diagnoses; a causal localization additionally requires a replay and repair intervention.

Run	Task	Recorded failure
R005	T1	fixed-point arithmetic
R034	T1	fixed-point arithmetic
R009	T2	cross-layer integration
R023	T2	cross-layer integration
R038	T2	recurrent state
R012	T3	recurrent state
R013	T3	recurrent state
R027	T3	recurrent state
R041	T3	cross-layer integration

Table 11: Reduced four-word state replay. The sole mutation suppresses the accepted write at step 2. State differs immediately, logits at step 4, and the selected token at step 6; restoring the update recovers every transition.

Step	Input	State differs	Logits differ	Token differs	Repair differs
0	1	0	0	0	0
1	2	0	0	0	0
2	3	1	0	0	0
3	4	1	0	0	0
4	5	1	1	0	0
5	4	1	1	0	0
6	4	0	1	1	0
7	8	0	0	0	0

I AN EXECUTABLE DELAYED-STATE FAILURE

A reduced integer transition makes the delay directly observable, instead of requiring a full-network failure to be inferred from a token trace. Let s_t contain four words, newest first. For input x_t, define

$$z_t = [6,\ x_t + s_t[3]], \qquad s_{t+1} = [x_t, s_t[0], s_t[1], s_t[2]], \tag{4}$$

with zero initial state and the lower token index selected on an exact tie. All eight transactions are accepted. The inputs are $[1, 2, 3, 4, 5, 4, 4, 8]$, and the only mutation is to suppress the state update at transaction 2. No value approaches an integer overflow boundary. The reference, mutant and repaired executions share identical inputs and initialization.

The mutant emits correct logits at transaction 2 but leaves $[2, 1, 0, 0]$ instead of $[3, 2, 1, 0]$. Its oldest buffered value first changes a logit at transaction 4, where the reference emits $[6, 6]$ and the mutant emits $[6, 5]$; both still select token zero. At transaction 6, $[6, 7]$ versus $[6, 6]$ changes the selected token. Restoring the single missing update recovers all eight output-and-state transitions.

Listing 8: Exact transition checks from the reduced Python replay. The event order shows why a current-token check misses the first faulty commit. The complete reference, mutant and repaired states are retained in the JSONL companion.

```
step=2 state_ref=[3,2,1,0] state_mut=[2,1,0,0]
       logits_ref=[6,3] logits_mut=[6,3] token=0/0
step=4 logits_ref=[6,6] logits_mut=[6,5] token=0/0
step=6 logits_ref=[6,7] logits_mut=[6,6] token=1/0
repair=restore_writeback complete_transition_matches=8/8
```

Verilog circuit code. The following SystemVerilog circuit implements the same four-word transition, with a parameter that omits only the write at accepted transaction 2. Logits are sampled before the rising edge and state after the edge. Its signed 17-bit addition accommodates the sum of two signed 16-bit words; the eight inputs remain well inside that range.

Listing 9: Complete four-word circuit. The mutation parameter changes one accepted state write; the arithmetic and selected-token rule remain fixed. This module specifies the reduced transition used in Figure 3a.

```
module delayed_state_core #(
  parameter bit OMIT_WRITE_AT_TWO = 1'b0
) (
  input logic clk, rst_n,
  input logic in_valid,
  output logic in_ready,
  input logic signed [15:0] in_x,
  output logic signed [16:0] logit0, logit1,
  output logic token,
  output logic [63:0] architectural_state
);
  logic signed [15:0] s0, s1, s2, s3;
  logic [31:0] accepted_index;
  assign in_ready = rst_n;
  assign logit0 = 17'sd6;
  assign logit1 = {in_x[15], in_x} + {s3[15], s3};
  // Lower token index wins ties. Outputs refer to the state before commit.
  assign token = logit1 > logit0;
  assign architectural_state = {s3, s2, s1, s0};
  always_ff @(posedge clk) begin
    if (!rst_n) begin
      s0 <= '0; s1 <= '0; s2 <= '0; s3 <= '0;
      accepted_index <= '0;
    end else if (in_valid && in_ready) begin
      accepted_index <= accepted_index + 1;
      if (!(OMIT_WRITE_AT_TWO && accepted_index == 2)) begin
        s0 <= in_x; s1 <= s0; s2 <= s1; s3 <= s2;
      end
    end
  end
endmodule
```

J ADDITIONAL DECODER FAMILIES AND LARGER CONFIGURATIONS

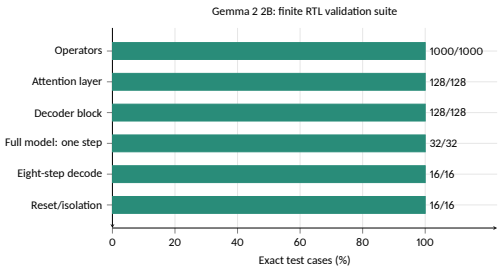
The additional target workloads are Gemma 2 2B (`google/gemma-2-2b`) and the text decoder of Gemma 3 4B (`google/gemma-3-4b-it`) (Google, 2024; Google DeepMind, 2025). Image processing is excluded from the Gemma 3 task. The GDN and convolution components of s_t in Equation 1 are empty for attention-only targets.

Both public suites allocate 1,000 cases to primitive operators, 128 to an attention layer, 128 to a complete decoder block, 32 to one-step model execution, 16 to eight-step decoding, and 16 to reset/isolation. The held-out suite allocates 16 cases each to new token prefixes, supported nonzero cache positions, reset after a nonempty history, bounded in-order stalls, and shape-compatible parameter reload. The four-workload comparison appears in Table 12. The Gemma-specific record below retains exact numerators for each validation level.

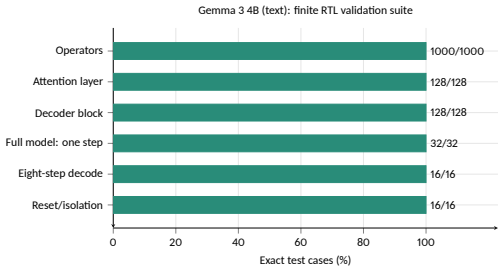
Table 12: Additional text-decoder checks, distinct from the nano agent cohort. Public counts use six validation levels; hidden counts use five families of 16 cases. Fidelity uses 1,600 common-prefix decisions. Qwen3.6-27B has integration failures despite exact layer checks.

Target workload	Public exact	Hidden exact	Top-1 matches	Fidelity (%)
Gemma 2 2B	1320/1320	80/80	1579/1600	98.69
Gemma 3 4B (text)	1320/1320	80/80	1588/1600	99.25
Qwen3.6-27B	1312/1320	70/80	1576/1600	98.50
Llama 3.2-1B	1320/1320	80/80	1589/1600	99.31

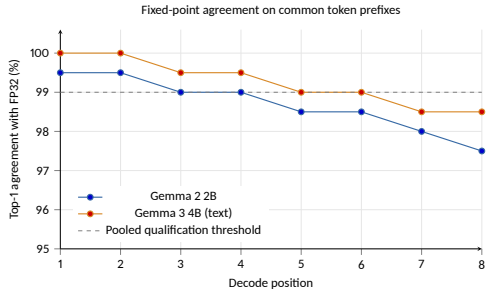
The fidelity comparison uses the same prefixes for FP32 and integer execution. Gemma 2 2B has 1,579 matching top-1 decisions out of 1,600, while Gemma 3 4B has 1,588. Gemma 2 2B therefore misses the 99% workload-qualification threshold even though its RTL matches its integer oracle. These additional reference passes broaden the tested transition contracts without increasing the number of independent agent successes. Figure 6c retains the per-position percentages rather than treating all token decisions as independent trials.



(a) Gemma 2 2B public reference checks. Each bar reports the fraction of exact cases at a validation level, with numerator and denominator printed at its end.



(b) Gemma 3 4B text-decoder reference checks with the same case allocation as Gemma 2 2B. The layer checks use the target architecture's own contract. Identical pass percentages do not establish identical test coverage or equal engineering difficulty across generations.

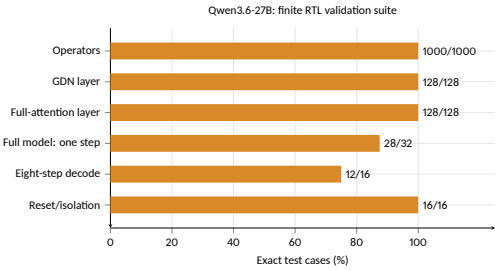


(c) Fixed-point fidelity on common token prefixes, using 200 decisions at each of eight decode positions. Pooled agreement is 98.6875% for Gemma 2 2B and 99.25% for Gemma 3 4B. The dashed 99% line denotes the pooled qualification threshold, not a requirement on each position. No independent-seed confidence interval is implied.

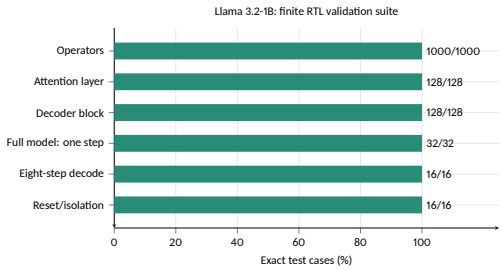
Figure 6: Gemma reference checks and fixed-point fidelity.

Qwen3.6-27B and Llama 3.2-1B. The Qwen3.6-27B configuration uses a dense hybrid decoder with 64 layers, hidden width 5,120, and a full-attention layer every fourth layer. Like the Qwen3.5 configurations, it enables a gated attention output (Qwen Team, 2026b;a;c). Reusing the Qwen3.5 scaffold still requires checking gate arithmetic, state layouts, and address ranges. The public allocation is 1,000 operator cases, 128 GDN-layer cases, 128 full-attention-layer cases, 32 one-step cases, 16 eight-step cases, and 16 reset cases. Operator and layer checks pass, but four one-step and four decode cases fail (Figure 7a). The hidden counts for new prefixes, cache positions, reset after history, bounded stalls, and idle reload are 14/16, 13/16, 16/16, 12/16, and 15/16, respectively.

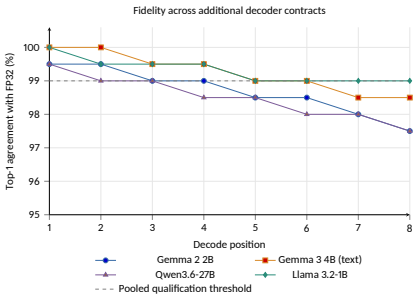
Llama 3.2-1B (Meta, 2024) supplies a compact, text-only autoregressive target from a third family. Its public allocation matches the Gemma cases, with its own attention, normalization, and positional contract, and all 1,320 public and 80 held-out checks pass (Figure 7b). The 1B target keeps a cross-family port bounded in scale, but its pass rate does not establish that other Llama configurations or longer contexts would pass. Neither the model name nor a reused matrix kernel guarantees an adaptation time. Figure 7c gives per-position fidelity for the four additional targets; pooled agreement is 1,579, 1,588, 1,576 and 1,589 of 1,600 decisions for Gemma 2 2B, Gemma 3 4B, Qwen3.6-27B and Llama 3.2-1B, so Gemma 2 2B and Qwen3.6-27B fall below the 99% criterion.



(a) Qwen3.6-27B reference checks. Exact operator and layer results coexist with 28/32 one-step and 12/16 eight-step passes.



(b) Llama 3.2-1B reference checks across six finite validation levels.



(c) Fidelity across the four additional targets on common token prefixes, with 200 decisions at each of eight positions. Pooled top-1 agreement is 1,579/1,600 for Gemma 2 2B, 1,588/1,600 for Gemma 3 4B, 1,576/1,600 for Qwen3.6-27B and 1,589/1,600 for Llama 3.2-1B. The dashed line is the pooled 99% workload criterion. The compared tokens are correlated within a sequence, and no independent-run uncertainty is inferred.

Figure 7: Qwen3.6-27B and Llama 3.2-1B reference checks, and fidelity for the four additional targets.

Interpreting terminal records. A case ledger records the workload, suite, case identifier, and complete-transition outcome. An ordered terminal trace summarizes these records at each validation boundary; it differs from a tool-call log, which additionally requires command arguments, matching invocation identifiers, exit codes, and elapsed time. Each of the four additional workloads has 1,400 ledger entries.

Listing 10: Terminal summaries for the additional decoder contracts.

```
Gemma-2-2B      public=1320/1320 hidden=80/80 fidelity=1579/1600
Gemma-3-4B-text public=1320/1320 hidden=80/80 fidelity=1588/1600
```

```
Qwen3.6-27B    public=1312/1320 hidden=70/80 fidelity=1576/1600
Llama-3.2-1B   public=1320/1320 hidden=80/80 fidelity=1589/1600
```

K ADDITIONAL PLOTS AND DIAGNOSTICS

Figures 8a–17b repeat one set of views for the three Qwen reference designs: Qwen3.5-nano (Figures 8a–9d), Qwen3.5-0.8B (Figures 10a–13b) and Qwen3.5-4B (Figures 14a–17b). They describe paired reference designs and, for 0.8B and 4B, one scale-adaptation trajectory each. They are not agent-cohort results, and each design is compared only with its own workload's baseline.

Correctness and fidelity. Every required public validation level passes for the three references (Figures 8a, 10a and 14a), and the held-out suites for 0.8B and 4B pass 232/232 and 288/288 cases (Figures 10b and 14b). Fixed-point agreement with FP32 declines across the eight decoded positions, from 99.7% to 98.8% for 0.8B and from 99.6% to 98.8% for 4B (Figures 10c and 14c). The final position is below 99% in both workloads, and position 7 is also below it for 4B, although pooled agreement meets the criterion. The decisions share token prefixes and are correlated.

Throughput, cycles and resources. The paired throughput, cycle and resource plots (Figures 8b–8d, 10d–11b and 14d–15b) give the per-workload throughput, cycle and LUT values behind Block B of Table 1. The longest post-synthesis path delays of the 0.8B and 4B designs are 7.68 and 7.82 ns against the 7.9365 ns period of the 126 MHz target, leaving about 0.26 and 0.12 ns (Figures 11c and 15c). In the cycle attributions (Figures 9c, 13a and 17a), the largest reduction for 0.8B and 4B is in external-memory stalls, from 7.5 to 5.9 and from 47.0 to 34.5 million cycles per token. The Qwen3.5-nano counters have no external-memory category; their largest reductions are in pipeline/control and state/cache movement. The nano optimization sequence (Figure 9a) raises throughput from 365.2 to 420 tokens/s in four schedule changes of 12 to 16 tokens/s each; it records one ordering and does not isolate each change's contribution. As the prompt grows, prefill cycles per prompt token fall and decode cycles per generated token rise in all three workloads, and the optimized schedule is lower at every tested length (Figures 9b, 12d and 16d). The LUT-budget plots (Figures 9d, 13b and 17b) compare separate resource configurations.

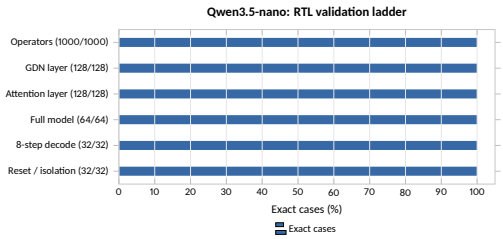
Scale-adaptation trajectories. For 0.8B and 4B, the trajectory, tool, clean-replay and gate plots (Figures 11d–12c and 15d–16c) expand Table 4 and Listing 3. The agent exits at 7.25 and 9.75 h and the blind evaluator finishes at 7.60 and 10.60 h; agent-side invocations total 168 and 210, with 11 and 17 failures; a clean replay from correct RTL takes 162.2 and 225.8 min. Each workload has one trajectory, so these plots document feasibility and cost accounting, not the reliability of scale adaptation.

Energy. Figures 18a–19b expand Section 6.2. With the FPGA-plus-DRAM power inputs, paired efficiency rises by 11.3%, 10.1% and 7.1% for Qwen3.5-nano, 0.8B and 4B. The sensitivity curves show that these gains would vanish if optimized power rose by 15% (nano and 0.8B) or 12.5% (4B) instead of 3.3–5.0%.

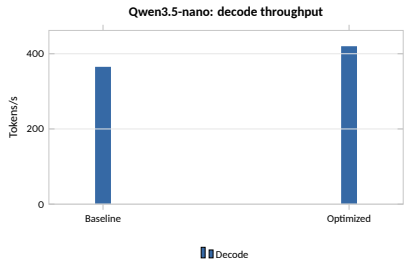
Nano-task cohort. Figures 20a–20d summarize the 45-attempt pilot in Block A of Table 1: public and final counts (13/15, 10/15 and 6/15 against 11/15, 7/15 and 2/15), final acceptance by level, primary failure labels and wall time per accepted run. The failure labels in Figure 20c cover every unsuccessful run (4, 8 and 13 per level), whereas Section 5 and Figure 5b analyze only the nine public-passing rejections; on T3, recurrent-state and reset labels account for 8 of the 13 failures. Wall time per accepted run rises with the level, and its numerator includes failed and timed-out attempts (Table 9).

L CODING AGENTS AND PUBLIC STATE OBSERVABILITY

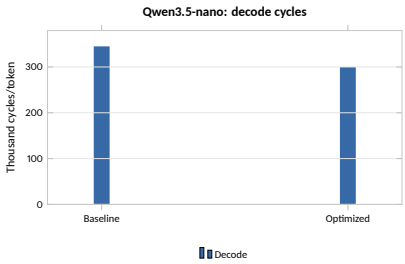
Experimental unit and controls. One attempt is a fresh agent process and workspace, ending at submission or a stopping limit. Each pair shares the task-instance bundle: initial RTL files, quantized parameter arrays, public inputs and the held-out assignment. The task definition and



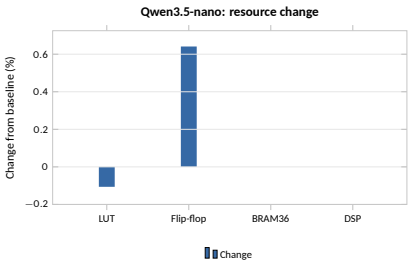
(a) Qwen3.5-nano: every required level of the public validation suite passes. Outputs, GDN recurrent state, convolution history and KV state are compared with the frozen fixed-point oracle at each relevant boundary.



(b) Qwen3.5-nano decode throughput at 126 MHz and prompt length eight. The baseline uses 345,000 cycles/token and the optimized design 300,000; speed increases from 365.22 to 420.00 tokens/s (15.0%). Precision and R3 tests are unchanged.

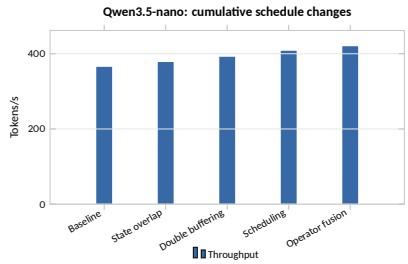


(c) Qwen3.5-nano cycles per decoded token under the same workload and fixed clock. The reduction is 13.04%, which yields a 15.0% reciprocal throughput increase, not a 15% cycle reduction.

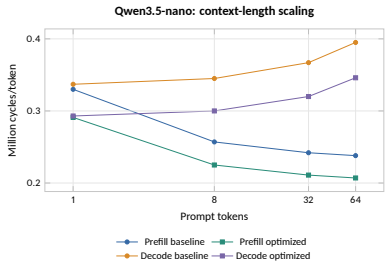


(d) Qwen3.5-nano resource changes at fixed device capacity. LUTs decrease from 47,100 to 47,050, flip-flops increase from 39,000 to 39,250, and BRAM36/DSP use remains 146/64.

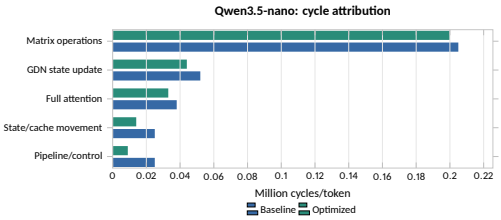
Figure 8: Qwen3.5-nano verification and paired hardware performance.



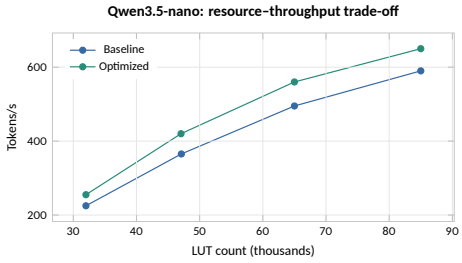
(a) Qwen3.5-nano cumulative optimization sequence: baseline, state overlap, double buffering, scheduling, and fusion.



(b) Qwen3.5-nano: average prefill cycles per prompt token and mean decode cycles per generated token, over eight generated tokens, for each prompt length. Increasing prompts amortize prefill setup in this range, while attention/cache work raises decode cost.

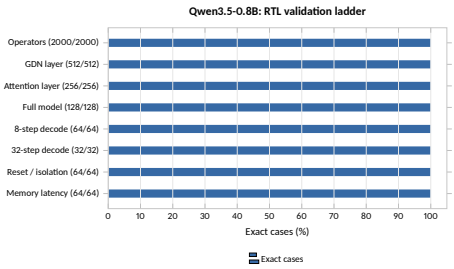


(c) Qwen3.5-nano: mutually exclusive cycle-owner counters for the same eight-token prompt used in the throughput comparison. The components sum to total elapsed cycles for each design. For overlapping execution, cycles are assigned once by a frozen priority rule; independently active utilization counters must not be summed this way.

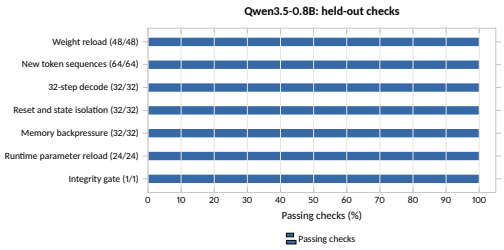


(d) Qwen3.5-nano: throughput versus LUT budget for separate configurations. The central pair corresponds to the canonical snapshot; other points are distinct resource configurations and are not claimed to have identical DSP/BRAM counts. A curve lying above another is an observed trade-off for these points, not a proven Pareto frontier or a cross-platform speedup.

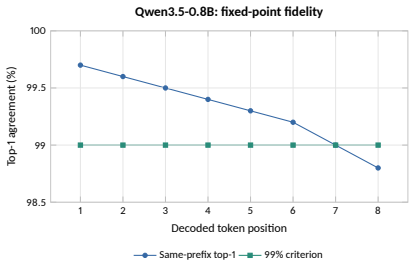
Figure 9: Qwen3.5-nano scheduling, context, and resource trade-offs.



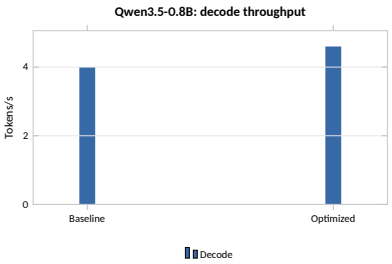
(a) Qwen3.5-0.8B: every required level of the public validation suite passes. Outputs, GDN recurrent state, convolution history and KV state are compared with the frozen fixed-point oracle at each relevant boundary.



(b) Qwen3.5-0.8B: held-out checks use new seeds and short, contract-valid sequences, including 32-step decoding and at most three stall cycles.



(c) Qwen3.5-0.8B: per-position top-1 agreement of fixed-point and FP32 logits using the same input prefix and paired evolving states.



(d) Qwen3.5-0.8B: batch-one decode throughput at 126 MHz, prompt length eight, frozen arithmetic, and identical memory-service model.

Figure 10: Qwen3.5-0.8B correctness and throughput.

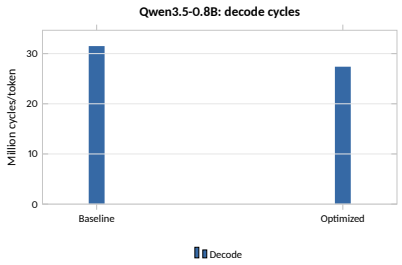
Table 13: Per-agent task allocation and common limits. The nano feedback comparison adds 16 output-only Sol attempts each for T2 and T3, reusing their output-and-state counterparts. The adaptation track supplies a shared scaffold; nano tasks supply their declared interface and starting files.

Task	Pairs	Wall h	Tool cap	Output k	Public	Hidden
Nano T1	16	3	180	90	1128	128
Nano T2	16	5	260	140	1320	64
Nano T3	16	6	320	180	1384	96
Qwen 0.8B	12	12	420	260	3120	232

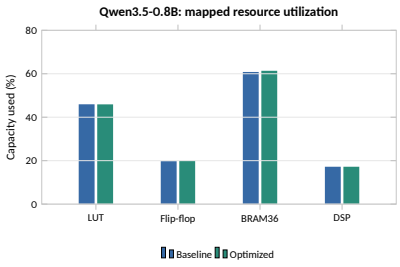
arithmetic remain fixed within each level. Pairing does not impose identical sampling trajectories or equate provider-specific reasoning compute. The same textual instructions and MathCode tool interface are used within a comparison. The display names identify the two agent configurations; runtime model identifiers and reasoning settings belong to the execution manifest.

Feedback intervention. The output-only condition receives complete-logit comparisons, interface/protocol outcomes and ordinary compiler diagnostics. Reference-state values, state-only failures and counters derived from state comparisons are withheld during construction. The output-and-state condition adds state checks and integer first-mismatch localization. In both conditions, the task specification describes the required state semantics, and agents may inspect their own RTL. The reference implementation is accessed through the permitted oracle API rather than exposed as readable source. Nano tasks have no additional local-formal feedback channel. After a submission is frozen, the grader performs the same full state-aware public recheck and hidden checks, so visible and canonical public success are reported separately (Table 5, Block B).

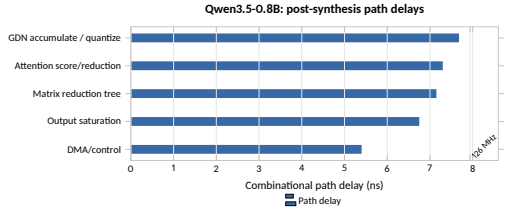
Sample allocation and reuse. The agent comparison contains 96 nano attempts and 24 adaptation attempts. The feedback experiment adds 32 output-only Sol attempts, bringing the number of unique



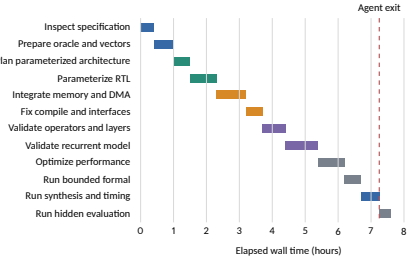
(a) Qwen3.5-0.8B: elapsed accelerator clock cycles per decoded token. Host/API latency, one-time weight upload, and agent design time are excluded. Multiplying by the 126 MHz clock period converts cycles to token latency.



(b) Qwen3.5-0.8B: LUT, flip-flop, BRAM36 and DSP use. On-chip buffers do not contain the full model weights; external memory is accounted for separately.

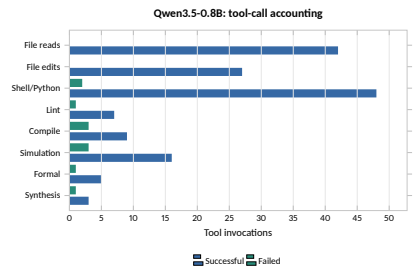


(c) Qwen3.5-0.8B: path-group delays, compared with the 7.9365 ns period of a 126 MHz target. The longest path is 7.68 ns.

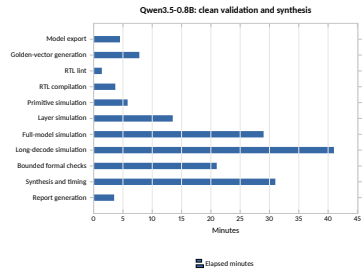


(d) Qwen3.5-0.8B: one scaffolded scale-adaptation trajectory. The agent exits at 7.25 h; a separate blind evaluator finishes at 7.60 h. The hidden phase is not feedback available to the agent. Phase blocks include reasoning, tool execution and inspection, so they are not exclusively simulator CPU time.

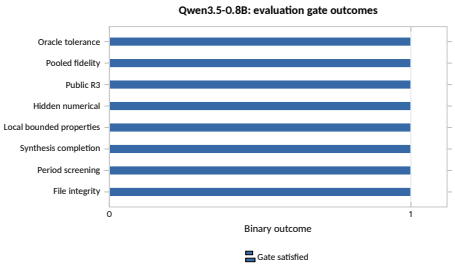
Figure 11: Qwen3.5-0.8B hardware and agent trajectory.



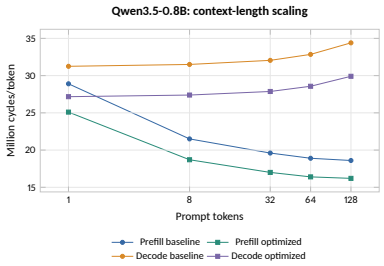
(a) Qwen3.5-0.8B: invocations grouped by tool type and outcome. Failed calls are included within each tool category rather than added as a ninth category. The total is 168, of which 11 failed. Agent calls exclude the final evaluator-only hidden phase.



(b) Qwen3.5-0.8B: one serial, fresh-build replay starting from frozen correct RTL. The total is 162.2 min, including 31 min for synthesis and timing. Public and held-out workloads are included once; hidden checks are not added a second time. Model download, agent generation/debugging, licenses, and queue delays are excluded.

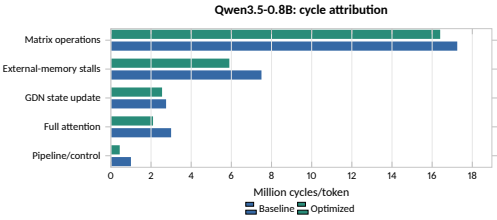


(c) Qwen3.5-0.8B: separate gate outcomes for this trajectory. Public and hidden gates require exact equality to fixed point. Local bounded properties concern controllers only.

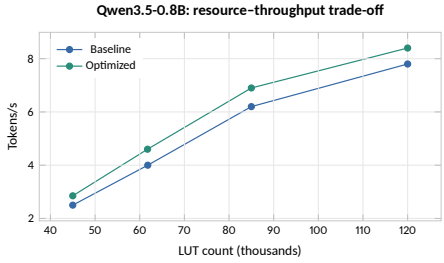


(d) Qwen3.5-0.8B: average prefill cycles per prompt token and mean decode cycles per generated token, over eight generated tokens, for each prompt length. Increasing prompts amortize prefill setup in this range, while attention/cache work raises decode cost. The x-axis is logarithmic in base two.

Figure 12: Qwen3.5-0.8B tools, replay, gates, and context.

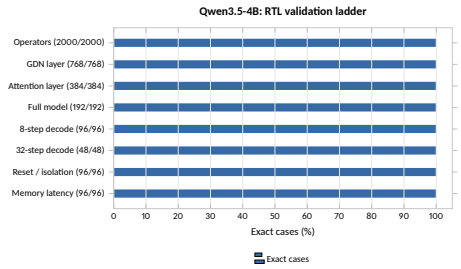


(a) Qwen3.5-0.8B: mutually exclusive cycle-owner counters for the same eight-token prompt used in the throughput comparison. The components sum to total elapsed cycles for each design. For overlapping execution, cycles are assigned once by a frozen priority rule; independently active utilization counters must not be summed this way.

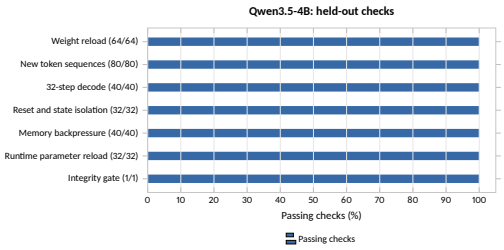


(b) Qwen3.5-0.8B: throughput versus LUT budget for separate configurations. The central pair corresponds to the canonical snapshot; other points are distinct resource configurations and are not claimed to have identical DSP/BRAM counts. A curve lying above another is an observed trade-off for these points, not a proven Pareto frontier or a cross-platform speedup.

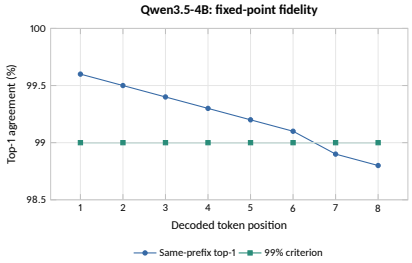
Figure 13: Qwen3.5-0.8B cycle attribution and resource trade-offs.



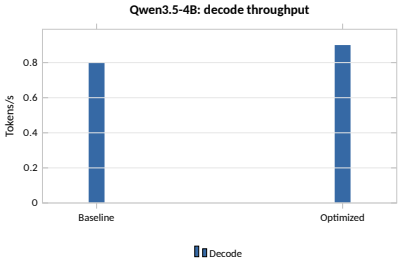
(a) Qwen3.5-4B: every required level of the public validation suite passes. Outputs, GDN recurrent state, convolution history and KV state are compared with the frozen fixed-point oracle at each relevant boundary.



(b) Qwen3.5-4B: held-out checks use new seeds and short, contract-valid sequences, including 32-step decoding and at most three stall cycles. Weight tests reload independently generated quantized arrays of supported shapes; they do not evaluate unseen trained model families.

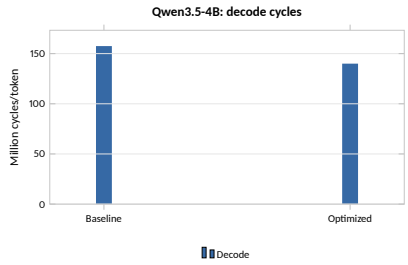


(c) Qwen3.5-4B: per-position top-1 agreement of fixed-point and FP32 logits using the same input prefix and paired evolving states. Each position contains 1,000 decisions; pooling therefore equals the arithmetic mean (99.2250%).

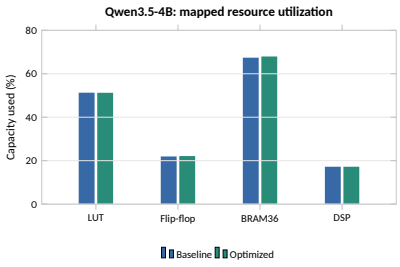


(d) Qwen3.5-4B: batch-one decode throughput at 126 MHz, prompt length eight, frozen arithmetic, and identical memory-service model.

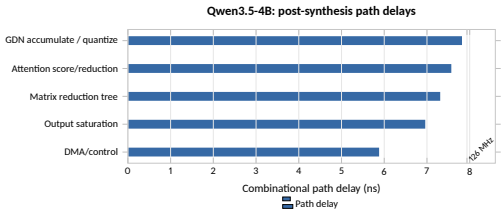
Figure 14: Qwen3.5-4B correctness and throughput.



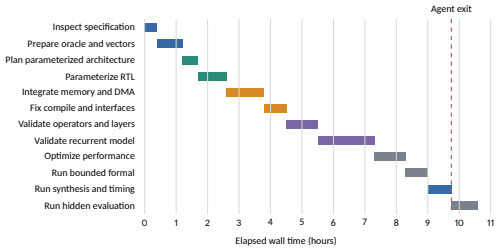
(a) Qwen3.5-4B: elapsed accelerator clock cycles per decoded token. Host/API latency, one-time weight upload, and agent design time are excluded. Multiplying by the 126 MHz clock period converts cycles to token latency.



(b) Qwen3.5-4B: LUT, flip-flop, BRAM36 and DSP use. On-chip buffers do not contain the full model weights; external memory is accounted for separately.

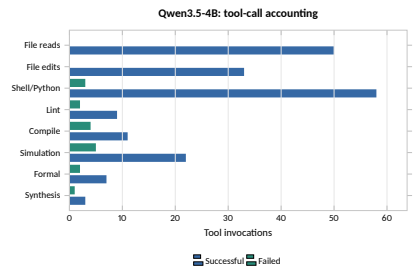


(c) Qwen3.5-4B: path-group delays, compared with the 7.9365 ns period of a 126 MHz target. The longest path is 7.82 ns.

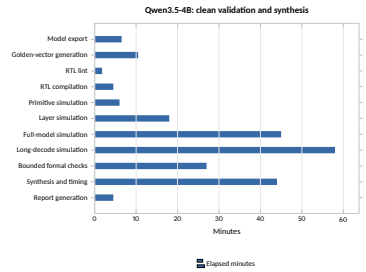


(d) Qwen3.5-4B: one scaffolded scale-adaptation trajectory. The agent exits at 9.75 h; a separate blind evaluator finishes at 10.60 h. The hidden phase is not feedback available to the agent. Phase blocks include reasoning, tool execution and inspection, so they are not exclusively simulator CPU time.

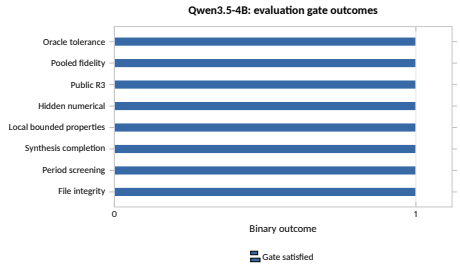
Figure 15: Qwen3.5-4B hardware and agent trajectory.



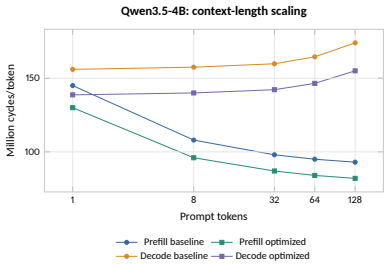
(a) Qwen3.5-4B: invocations grouped by tool type and outcome. Failed calls are included within each tool category rather than added as a ninth category. The total is 210, of which 17 failed. Agent calls exclude the final evaluator-only hidden phase.



(b) Qwen3.5-4B: one serial, fresh-build replay starting from frozen correct RTL. The total is 225.8 min, including 44 min for synthesis and timing. Public and held-out workloads are included once; hidden checks are not added a second time. Model download, agent generation/debugging, licenses, and queue delays are excluded.

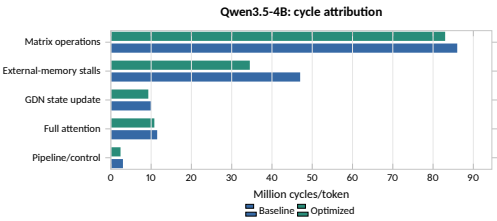


(c) Qwen3.5-4B: separate gate outcomes for this trajectory. Public and hidden gates require exact equality to fixed point.

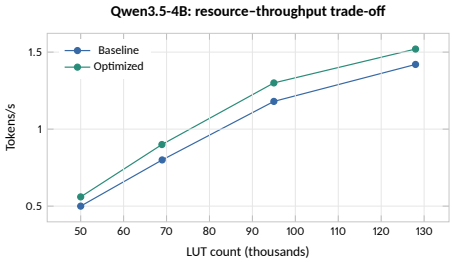


(d) Qwen3.5-4B: average prefill cycles per prompt token and mean decode cycles per generated token, over eight generated tokens, for each prompt length. Increasing prompts amortize prefill setup in this range, while attention/cache work raises decode cost. The x-axis is logarithmic in base two.

Figure 16: Qwen3.5-4B tools, replay, gates, and context.

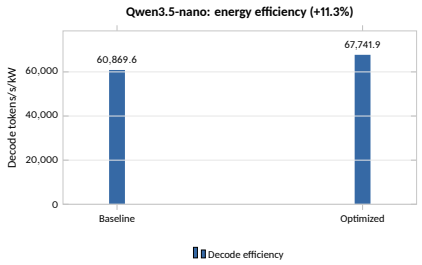


(a) Qwen3.5-4B: mutually exclusive cycle-owner counters for the same eight-token prompt used in the throughput comparison. The components sum to total elapsed cycles for each design. For overlapping execution, cycles are assigned once by a frozen priority rule; independently active utilization counters must not be summed this way.

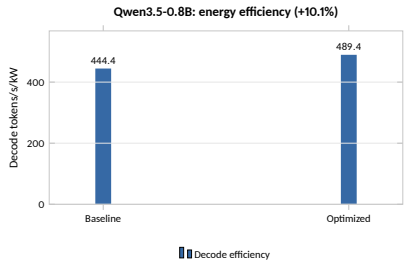


(b) Qwen3.5-4B: throughput versus LUT budget for separate configurations. The central pair corresponds to the canonical snapshot; other points are distinct resource configurations and are not claimed to have identical DSP/BRAM counts. A curve lying above another is an observed trade-off for these points, not a proven Pareto frontier or a cross-platform speedup.

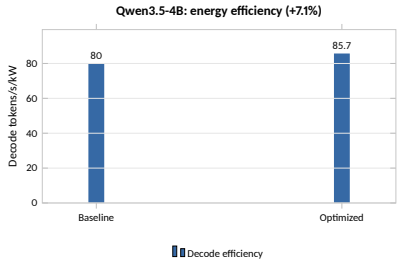
Figure 17: Qwen3.5-4B cycle attribution and resource trade-offs.



(a) Qwen3.5-nano: paired decode efficiency under the FPGA-plus-DRAM boundary. Baseline power is 6.0 W and optimized power 6.2 W. Efficiency rises from 60,869.57 to 67,741.94 tokens/s/kW (11.3%).

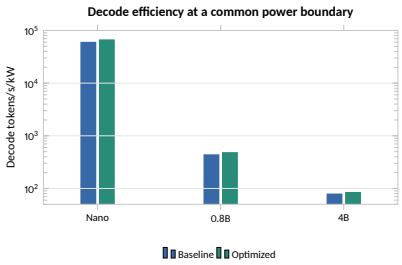


(b) Qwen3.5-0.8B: paired decode efficiency under the FPGA-plus-DRAM boundary. Baseline power is 9.0 W and optimized power 9.4 W. Efficiency rises from 444.44 to 489.36 tokens/s/kW (10.1%).

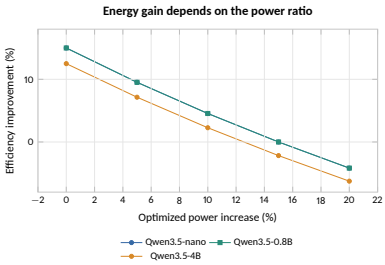


(c) Qwen3.5-4B: paired decode efficiency under the FPGA-plus-DRAM boundary. Baseline power is 10.0 W and optimized power 10.5 W. Efficiency rises from 80.00 to 85.71 tokens/s/kW (7.1%).

Figure 18: Paired energy efficiency by workload.

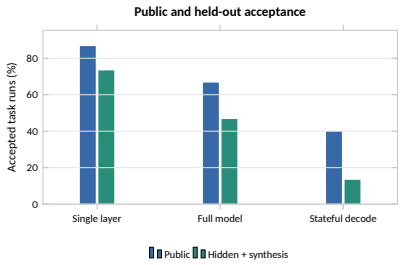


(a) Decode efficiency under the FPGA-plus-DRAM power boundary. Bars use 6.0/6.2 W (nano), 9.0/9.4 W (0.8B), and 10.0/10.5 W (4B) for baseline/optimized designs. The logarithmic y-axis allows different model sizes to remain visible; only paired within-model ratios are comparable. Values equal 1,000 times tokens/joule and are not mixed-token serving throughput.

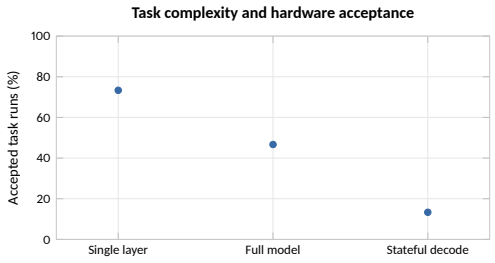


(b) Efficiency sensitivity. Each curve divides the throughput ratio by the optimized-to-baseline power ratio. Efficiency reaches break-even when power grows by 15% for nano/0.8B or 12.5% for 4B; throughput improvements alone do not establish an energy benefit.

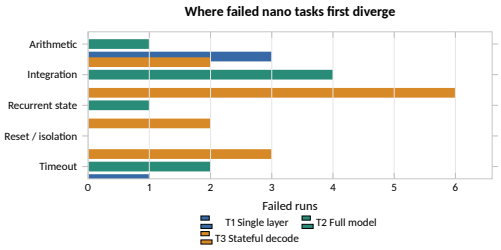
Figure 19: Energy boundary and power sensitivity.



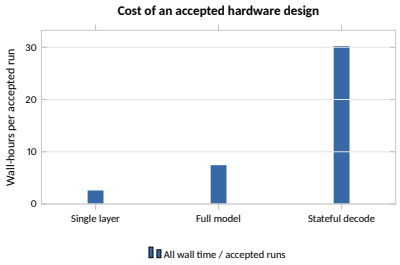
(a) Fifteen nano-task runs per difficulty level, with five runs for each of three agent-system configurations. Counts are 13/15, 10/15 and 6/15 on public tests versus 11/15, 7/15 and 2/15 after held-out and synthesis gates. Different difficulty levels are separate tasks, not successive checkpoints of the same run.



(b) Hardware acceptance by task difficulty, with 11/15, 7/15 and 2/15 passing runs. The workload and arithmetic stay fixed while cross-layer and temporal obligations increase.



(c) One primary failure label per unsuccessful run. Arithmetic failures dominate the single-layer task; recurrent state and reset account for eight of thirteen failures on stateful decode.



(d) Total wall time of all 15 attempts divided by accepted runs at each difficulty. Failed attempts and six-hour timeouts remain in the numerator.

Figure 20: Nano-task acceptance, failure localization, and cost.

Table 14: Within-instance contrasts. Left/right-only counts are discordant accepted outcomes. Differences are in percentage points; the two-sided exact McNemar test conditions on discordant pairs. Pairing controls the assigned instance, not language-model randomness. These exploratory comparisons do not establish a model ranking.

Contrast	Pairs	Left	Right	Diff.	p
A−S: Nano T1	16	2	1	-6.2	1.000
A−S: Nano T2	16	2	3	+6.2	1.000
A−S: Nano T3	16	1	3	+12.5	0.625
A−S: Qwen 0.8B	12	1	2	+8.3	1.000
State−output: T2	16	1	3	+12.5	0.625
State−output: T3	16	1	3	+12.5	0.625

Table 15: Construction costs over all attempts. Tokens are generated output tokens in thousands; wall time is the per-run median, calls are means, and hours/accepted uses total wall time including failures. Grader time is separate.

Task	Agent	Median h	Calls	Failed	Output k	h/accepted
Nano T1	Sol 5.6	1.98	79.3	4.8	23.1	2.32
Nano T1	Astra 6	1.79	78.9	4.7	25.6	2.48
Nano T2	Sol 5.6	2.72	92.7	5.6	36.3	4.76
Nano T2	Astra 6	3.14	97.1	5.4	44.0	4.91
Nano T3	Sol 5.6	4.63	114.7	5.4	58.3	15.14
Nano T3	Astra 6	4.50	110.5	4.9	60.4	10.09
Qwen 0.8B	Sol 5.6	7.93	165.8	6.4	108.1	14.53
Qwen 0.8B	Astra 6	7.79	160.7	6.4	106.0	12.20

attempts to 152. Its output-and-state controls are the same 32 Sol T2/T3 attempts used in the agent comparison. Reuse is explicit and contributes no additional independent runs. The earlier 45-attempt pilot and the original 4B trajectory remain separate.

Acceptance and stopping. Final acceptance requires the canonical public suite, hidden numerical suite, synthesis and integrity, plus local checks for 0.8B. Timeouts count as failed attempts. Agent wall time ends at the frozen submission; grader time begins afterward. Tool counts include failed invocations and exclude post-exit grader work. The hidden evaluator supplies no feedback to the agent. Table 13 gives three simultaneous resource limits (wall time, tool calls, and output tokens); a run stops when any is reached.

Cost and hardware. Mean calls and output tokens include unsuccessful attempts. The empirical cost per accepted submission is the sum of all agent wall times divided by the number accepted, not median runtime divided by a pass fraction. Token counts are resource-use measures rather than a currency or a measure of equal reasoning compute. Hardware is described only for accepted 0.8B submissions, at 126 MHz, batch one and an eight-token prompt under the common memory-service assumptions. Different acceptance sets create selection effects, so the hardware scatter (Figure 21b) is descriptive rather than a paired estimate of a model's hardware-optimization skill.

Figure 21 adds four views of the same cohorts. Panel (a) places each 0.8B attempt by agent time and final outcome. Panel (c) shows that state/protocol is the most frequent primary diagnosis among public-passing rejections for both agents (four each); the counts pool nano T1–T3 and 0.8B and serve failure inspection only. Panel (d) breaks down the mean tool calls per 0.8B attempt, which total 165.8 for Sol and 160.7 for Astra (Table 15).

Listing 11: Accepted 0.8B submission and isolated final grading. Times are elapsed seconds; counters reconcile with the run ledger.

```
run_id=sol_Q08_os_02
agent=Sol 5.6; harness=MathCode
task=Q08; pair=Q08-02; feedback=output_state
agent_exit_s=27533; termination=submitted
tool_calls=159; failed_calls=6
input_tokens=423953; output_tokens=104909
visible_public=3120/3120
freeze; agent_access=closed; grader_feedback_to_agent=false
canonical_public=3120/3120
hidden_numerical=232/232
synthesis=true; integrity=true
local_required=true; local_exact=36/36
grader_exit_s=28577
accepted=true; outcome=accepted
target_clock_hz=126000000; cycles_per_token=28938780
tokens_per_second=4.354019; LUT=61690
FF=53370; BRAM36=222; DSP=128
```

Listing 12: Accepted 0.8B submission and isolated final grading. Times are elapsed seconds; counters reconcile with the run ledger.

```
run_id=astra_Q08_os_01
agent=Astra 6; harness=MathCode
task=Q08; pair=Q08-01; feedback=output_state
agent_exit_s=27480; termination=submitted
tool_calls=149; failed_calls=10
input_tokens=329106; output_tokens=92333
visible_public=3120/3120
freeze; agent_access=closed; grader_feedback_to_agent=false
canonical_public=3120/3120
hidden_numerical=232/232
synthesis=true; integrity=true
local_required=true; local_exact=36/36
grader_exit_s=28618
accepted=true; outcome=accepted
target_clock_hz=126000000; cycles_per_token=29413206
tokens_per_second=4.283790; LUT=61640
FF=53730; BRAM36=226; DSP=128
```

Listing 13: Output-only T3 submission: visible public passing and the state-aware post-exit recheck give different outcomes. The agent receives no grader feedback.

```
run_id=sol_T3_oo_04
agent=Sol 5.6; harness=MathCode
task=T3; pair=T3-04; feedback=output_only
agent_exit_s=18417; termination=submitted
tool_calls=118; failed_calls=7
input_tokens=309874; output_tokens=75993
visible_public=1384/1384
freeze; agent_access=closed; grader_feedback_to_agent=false
canonical_public=1378/1384
hidden_numerical=95/96
synthesis=true; integrity=true
local_required=false; local_exact=0/0
grader_exit_s=18687
accepted=false; outcome=public_integration
```

Table 16: Qwen3.5-0.8B hardware among accepted submissions. Throughput uses the common target clock and memory assumptions, and LUT/BRAM values are medians. Accepted counts differ between agents, so these descriptive hardware distributions do not replace the unconditional reliability comparison.

Agent	Accepted	Median tok/s	Range	Median LUT	BRAM	DSP
Sol 5.6	7	4.398	4.240–4.494	61,995	226	128
Astra 6	8	4.337	4.225–4.527	61,910	224	128

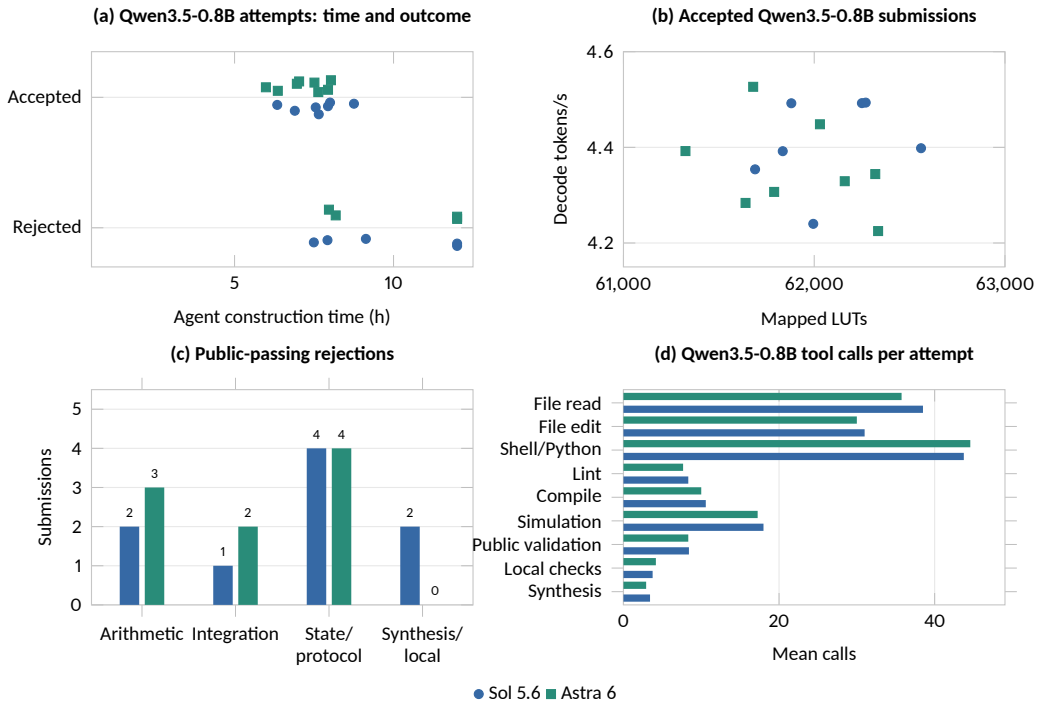


Figure 21: Agent-comparison diagnostics under output-and-state feedback. (a) Each point is one Qwen3.5-0.8B attempt: agent wall time before the submission was frozen, excluding the separate grader duration, and the final outcome. Small vertical offsets separate agents and coincident outcomes; rejections and 12-hour wall-limit terminations remain in the plot. (b) Mapped LUTs and target-clock decode throughput (126 MHz divided by elapsed cycles per token, at batch one and an eight-token prompt, with a shared memory-service model) for the seven accepted Sol and eight accepted Astra 0.8B submissions, all of which passed the same final gates. The acceptance sets differ, so this panel is descriptive rather than a paired estimate of hardware-optimization skill. (c) Primary diagnoses among public-passing rejections, pooled over nano T1–T3 and 0.8B only for failure inspection. Categories are mutually exclusive: arithmetic, integration, and state/protocol labels denote hidden numerical rejections, and synthesis/local labels denote those gates. Feedback-ablation additions are excluded, and not every rejection is interpreted as a delayed-state failure. (d) Mean tool invocations per 0.8B attempt over all 12 attempts per agent; failed invocations stay in their original categories, and post-exit grader calls are excluded.