CHIPFACTORY: CAPTURING THE WORLD OF CHIP DESIGN AT SCALE

Anonymous authors
Paper under double-blind review

ABSTRACT

Advancing AI for chip design requires models that can understand complex hardware, optimize designs, and assess their functional and implementation outcomes. Studying these capabilities calls for data that connect diverse design contexts and implementation choices with observable EDA backend feedback. However, previous work rarely combines design complexity, diversity, and backend feedback richness at scale. We present ChipFactory, a dataset of over 1 million design records covering complex hardware across diverse sources, abstraction levels, and implementation choices, with rich backend feedback from simulation through physical implementation. ChipFactory combines designs from generators and compilers, external datasets, human-designed accelerators, and software programs, relying on HLS together with AI agents to expand the latter two into design variants at scale. Reusable agent skills and an open harness let researchers build their own benchmarks from these design records. We instantiate 13 tasks spanning implementation prediction, behavioral prediction, ranking, and generation, and evaluate twelve LLMs, from state-of-the-art proprietary models to open-source and domain-specialized ones. Even the strongest model, Claude Fable 5.1, achieves an average score of only 58.4 out of 100, and the four smallest open-source models solve none of the generation problems. ChipFactory is open source to help the community expose the challenges models still face in reasoning across hardware representations and train specialized models for chip design.

1 INTRODUCTION

With the great success of Large Language Models (LLMs) in general coding, there is growing interest in improving and evaluating AI for chip design. While many chip design tasks are essentially coding tasks with verifiable feedback, they present additional challenges, including limited high-quality open-source data and sparse feedback.

We argue that a comprehensive dataset for AI for chip design should address three dimensions. (1) *Design complexity* — SoTA LLM-based agents already achieve strong results on unit-level register-transfer level (RTL) and High-Level Synthesis (HLS) benchmarks (Ding et al., 2026a;b; Deng et al., 2026; Eryilmaz, 2026; Yu et al., 2026a), and the next challenge is large, structurally complex chip designs. (2) *Diversity of design context* — A dataset should cover different computation, source representations, and implementation choices. (3) *Rich backend feedback* — A dataset should connect designs to feedback from verification, synthesis, and physical implementation, so that agents can see the effects of their design choices.

Among the 69 previous works we surveyed, most released RTL designs are single modules (Liu et al., 2023; Wang et al., 2026a). Only 18 span multiple abstraction levels. Just 9 record routed results, while 40 record no backend feedback (Pinckney et al., 2025; Ghorab et al., 2025). CircuitNet 3.0 (Wang et al., 2026b) connects RTL, netlists, and layouts but does not target architecture-level optimizations such as parallelism, pipelining and memory organization. Previous HLS work (Bai et al., 2023; Abi-Karam et al., 2024; Peng et al., 2025) explores these optimizations but mainly sweeps pragmas on HLS-ready C/C++ kernels, leaving open-source software largely unexplored. To our knowledge, no previous work addresses all three dimensions at scale (Section 2).

We present CHIPFACTORY, a fully open dataset of over **1 million** design records with an average of **36K** lines of RTL. The records span complex designs and diverse design contexts, with rich feedback

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

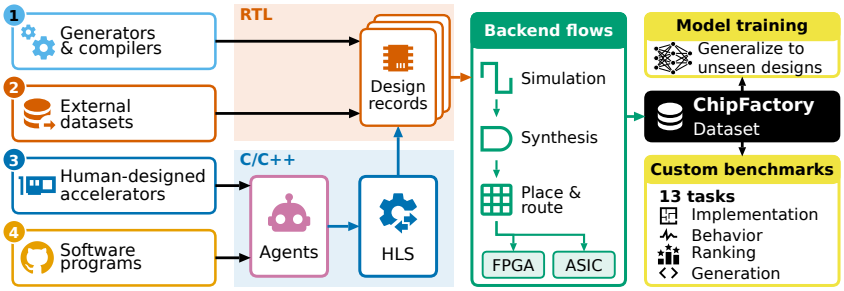


Figure 1: Overview of ChipFactory. Generators, compilers, and external datasets supply register-transfer level (RTL) designs directly, while AI agents and high-level synthesis (HLS) turn human-designed accelerators and software programs into RTL design variants. Each design record then goes through simulation, synthesis, and place and route for FPGA or ASIC targets, and the resulting dataset supports model training and custom benchmarks.

from multiple backend stages. ChipFactory collects designs from 32 generators and compilers and 38 external hardware datasets, and its source programs, RTL designs, and backend artifacts total 14.9 TB. We hope that ChipFactory can play a role in AI for chip design similar to the one ImageNet played in deep learning for image recognition (Deng et al., 2009).

The key to ChipFactory's methodology (Figure 1) for scaling complex chip designs is to combine four design sources: (1) generators and compilers, (2) external datasets, (3) human-designed accelerators, and (4) software programs. We rely on HLS together with AI agents to expand the latter two into design variants at scale. Open-source chip designs are scarce, but software is abundant. With HLS, agents can design chips by writing C++, which in turn is compiled into RTL and then passed to backend implementation tools for rich backend feedback (Section 3.2).

To demonstrate the flexibility and breadth of ChipFactory, we build a custom benchmark of 13 tasks and evaluate 12 commercial, open-source, and domain-specialized models (Section 5). Even the strongest model (Claude Fable 5.1 (Anthropic, 2026)) achieves an average score of only 58.4 out of 100. We also show that training a small supervised model on ChipFactory can outperform every LLM we evaluate in predicting frequency.

In summary, our contributions are:

- **Dataset.** We introduce ChipFactory, an open collection of over **1 million** design records with an average of **36K** lines of RTL, diverse design variants, and rich backend feedback.
- **Data generation pipeline.** We provide an agent-driven data generation pipeline, a reusable agent harness, and an evaluation framework, demonstrated through **13** tasks.
- **Evaluation and training.** We evaluate **12** commercial, open-source, and domain-specialized models, and train supervised models that generalize to unseen base designs.
- **Comparison with previous work.** We examine **69** previous works against their released artifacts, claimed results, and actual availability.

2 Comparison with Previous Work

We compare previous work by design complexity, design diversity, and backend feedback richness. Figure 2 summarizes 69 previous works; Table 1 compares nine representative previous works with ChipFactory. Appendix A provides the comparison details and backend availability definitions.

Design complexity. We measure design complexity by RTL lines of code (LOC) and module hierarchy depth. Large design counts do not imply large designs: PyraNet's 692,238 designs average 79 RTL lines, while ForgeHLS's measured designs average 4,669 (Nadimi et al., 2024; Peng et al., 2025). ChipFactory averages 36,068 RTL lines per design, about 8× the ForgeHLS mean. Hierarchy is also limited in previous work: 25 of the 57 works with measured RTL have no designs at depth ≥ 5 (Figure 2a). ChipFactory has 451,294 of its 1,006,725 designs (44.8%) at this depth, compared with 17,509 of 429,279 measured ForgeHLS designs (4.1%). ChipFactory thus provides a large collection of hierarchical designs for studying dependencies across modules.

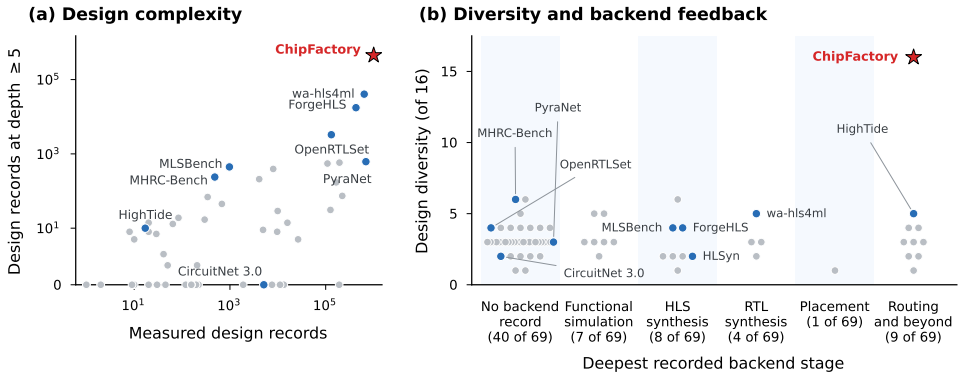


Figure 2: Comparison with previous work. (a) Design records whose released RTL we could measure, versus those with hierarchy depth ≥ 5 (57 previous works). (b) Number of the 16 diversity fields (abstraction level and target platform) that a work releases, versus the deepest backend stage whose results it releases for all its designs (69 previous works). Blue marks the previous works of Table 1; red stars mark ChipFactory, whose diversity count includes planned fields.

	Dataset size	Complexity		Diversity		Feedback richness
Work	#design	Mean RTL LOC	Depth ≥ 5	Abstraction levels	Target platforms	Recorded backend stages
MHRC-Bench (Zou et al., 2026)	584	146,842	238	HLS, RTL	General RTL	Not released
CircuitNet 3.0 (Wang et al., 2026b)	5,097	91	0	RTL, netlist	ASIC	RTL synth, Route
HighTide (Goldblatt et al., 2026)	16	60,370.2	10	RTL	ASIC	Route
wa-hls4ml (Hawks et al., 2026)	684,063	46,257.6	40,530	HLS, RTL	FPGA	HLS synth, RTL synth
ForgeHLS (Peng et al., 2025)	459,850	4,669	17,509	HLS, RTL	FPGA	HLS synth
OpenRTLSet (Wang et al., 2025)	131,309	90	3,280	NL, RTL	General RTL	Not released
PyraNet (Nadimi et al., 2024)	692,238	79	617	NL, RTL	General RTL	Compile
HLSyn (Bai et al., 2023)	41,596	N/A	N/A	HLS	FPGA	HLS synth
MLSBench (Goswami et al., 2022)	3,288	10,103	446	HLS, RTL	FPGA	C sim, RTL sim, HLS synth, RTL synth, Place, Route
ChipFactory (Ours)	**1,006,725**	**36,068**	**451,294**	NL, HLS, HCL, RTL	ASIC/FPGA	C sim, RTL sim, HLS synth, RTL synth, Place, Route

Table 1: Comparison with previous work. #design follows each work's counting unit; Mean RTL LOC averages the measured design records and Depth ≥ 5 counts qualifying measured design records. NL denotes natural language; N/A indicates no released RTL. Feedback lists stages with released tool results for individual design records; coverage may be partial (Appendix A.3).

Design diversity. Figure 2b summarizes diversity in abstraction levels and target platforms, with a detailed breakdown in Appendix A.2. Previous work often focuses on specific combinations, such as HLS-to-RTL pairs for FPGAs in ForgeHLS (Peng et al., 2025). ChipFactory combines designs from external datasets, generators and compilers, human-designed accelerators, and software programs. It reaches RTL through HLS tools such as Vitis HLS, Bambu, Catapult, and XLS, as well as hardware construction languages (HCLs) such as Chisel. Its target platforms range from AMD Xilinx, Altera, and Microchip FPGAs to the SkyWater 130, NanGate45, and ASAP7 ASIC technologies.

Backend feedback. Previous work often provides backend measurements with limited access to the intermediate artifacts behind them (Figure 2b). HLSyn provides HLS synthesis measurements without releasing the corresponding RTL (Bai et al., 2023). wa-hls4ml includes RTL synthesis measurements but does not release placement or routing results (Hawks et al., 2026). HighTide reports post-route area, timing and power, while intermediate synthesis and placement outputs require running its released flow (Goldblatt et al., 2026). MLSBench retains physical implementation reports or checkpoints for only six designs (Goswami et al., 2022). ChipFactory preserves simulation traces, module-level synthesis reports, timing paths and implementation checkpoints across

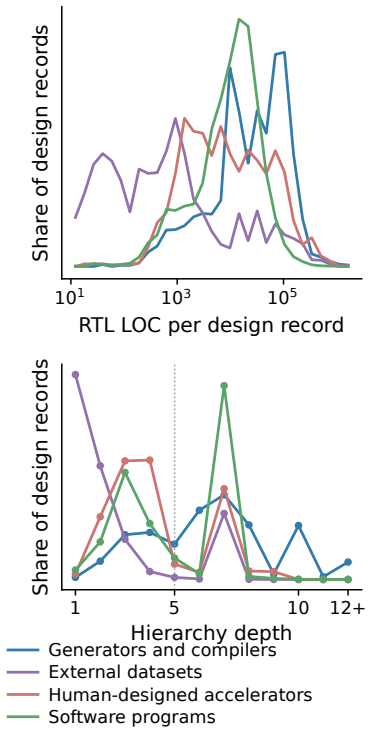


Figure 4: Design complexity by source class. Distributions of RTL lines of code (top) and hierarchy depth (bottom) per design record. The dotted line marks a depth of 5.

Source class	#Work / #Base / #Variant
Generators and compilers	32 / 1,169 / 341,928
External datasets	38 / 104,719 / 256,670
Human-designed accelerators	328 / 5,938 / 247,654
Software programs	14,429 / 14,429 / 160,473
Total	14,827 / 126,255 / 1,006,725

Frontend abstraction levels

Natural-language specifications; software programs and models; DSLs; HLS (Vitis HLS, Bambu, Catapult, XLS, Altera HLS 2026, Dynamatic, and SmartHLS); HCLs (Chisel, MyHDL, Amaranth, Migen, PyMTL, and PyRTL); RTL (Verilog, SystemVerilog, and VHDL).

Target platforms

AMD Versal HBM, Altera Stratix V, and Microchip SmartFusion2 and PolarFire FPGAs; generic FPGAs through VTR; NanGate45, SkyWater 130, and ASAP7 ASIC technologies.

Backend stage	Retained artifacts	#Variant
Simulation	Simulation logs, co-simulation reports, test vectors, waveforms, measured latency	86,337
HLS synthesis	Generated RTL, scheduling and binding data, module-level latency, resource and timing reports	739,053
RTL synthesis	Netlists, checkpoints, hierarchical utilization and timing reports	428,661
Placement	Checkpoints, utilization, timing, power and congestion reports	152,164
Routing	Checkpoints, utilization, timing paths, power, wirelength and congestion reports	50,834

Table 2: ChipFactory data coverage. We count the works, base designs, and design variants of each source class, where a work is the origin of its base designs. Each software program counts as one work and one base, and accelerator bases also include submodules. For simulation, we count design variants with waveforms, and for the other backend stages, we count design variants that complete the stage (full counts in Appendix B.2).

completed stages, linking them by design and configuration with the goal of giving users access to rich feedback that hardware designers use to evaluate design choices.

3 The ChipFactory Dataset

ChipFactory covers all three dimensions where previous works fall short (Section 2). We first describe the design record (Section 3.1), then explain how we scale up the dataset (Section 3.2), and finally report the dataset statistics (Section 3.3).

3.1 Unified Data Structure

We record each design variant in one directory, which we call a *design record*. The design record keeps the information a hardware designer produces and uses along the design flow, and links source code and RTL to the results of simulation, synthesis, and physical implementation. We show the design record of a matrix-multiplication (`gemm`) accelerator generated by Stream-HLS (Basalama & Cong, 2025) in Figure 3, where `index.json` tracks its origin and `status.json` tracks the result of each stage.

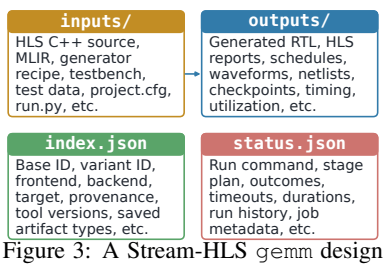


Figure 3: A Stream-HLS `gemm` design record (Appendix B.1).

3.2 Data Collection and Variant Generation

A *base design* is an original design, such as a software kernel, an accelerator or a processor. *Design variants* are produced by optimizing a base design. Beyond

collecting designs from external datasets (listed with the generators and compilers in Appendix B.1), we build hardware from software and also leverage tool-generated designs. With HLS, we can derive RTL directly from C++. The software program already defines the correct behavior, so it can serve as the golden model for the designs derived from it.

For software programs, we collect open-source programs from repositories and also use LLMs to write software specifications that fill the gaps of the repository programs. We then use agents to refactor each program into synthesizable HLS as a base design and optimize it into design variants that differ in parallelism, pipelining, resource sharing, and memory organization, which in turn are compiled by HLS into correct-by-construction RTL.

For human-designed accelerators, we use agents to rewrite their submodules and then sweep pragma configurations. We store the original source and the agent's rewrite separately, so researchers can tell how each design variant was produced.

From generators and compilers, we also collect complete systems such as Chipyard SoCs. We produce their design variants by changing workload dimensions, data types, transformation settings, hardware parameters, and clock targets in their input configurations.

We run FPGA and ASIC flows to different stages, and use clustering to select representative design variants of each base design for the later stages. Each design record keeps the artifacts of each completed stage, from which we extract resource use, timing, power, and latency as labels.

3.3 Dataset Statistics

We discuss the statistics of ChipFactory's base designs and design variants. We further group base designs by "work", which is the generator or compiler, external dataset, original accelerator or original program they come from. ChipFactory contains 1,006,725 design records with available RTL, derived from 126,255 base designs. We list the works, base designs, and design variants of each source class in Table 2, together with the available representations, platforms, and backend artifacts.

Design complexity. The mean RTL size is 36,068 lines per design record, and 45% of design records reach a hierarchy depth of at least 5. Notably, our agent-driven scaling yields designs with high line counts and deep hierarchies, far beyond those in external datasets (Figure 4). At the median, designs from software programs have 11,257 RTL lines and designs from human-designed accelerators have 7,490, compared with 471 for designs from external datasets. Among designs from software programs, 51.8% reach a depth of 5, compared with 29.8% for human-designed accelerators and only 15.9% for external datasets (Appendix B.2). Generators and compilers contribute the largest designs, with a median of 28,026 lines.

Design diversity. The collection spans numerical kernels, neural networks, sorting, hashing, and cryptography, as well as processors and memory systems. External datasets contribute many base designs, but most of them are small, with a median of 471 RTL lines and 2 modules. In our agent-driven scaling, we use agents to refactor 14,429 software programs into new base designs and then optimize each into a median of 10 design variants. For each human-designed accelerator, we also use agents and configuration sweeps to produce a median of 461 design variants (Table 2). The design records also span frontend representations from natural-language specifications to RTL, and targets across AMD, Altera, and Microchip FPGAs and three ASIC PDKs (Table 2).

Backend feedback. We summarize the retained artifacts and stage coverage in Table 2, from signal-level simulation traces and module-level synthesis reports to physical implementation checkpoints and timing paths. Fewer design variants reach placement and routing, and we count the total, successful, and failed runs of each stage in Appendix B.2. In total, 85,452 design records have hierarchy depth ≥ 2, keep both the C++ or HCL source and the RTL it produces, and include placement or routing feedback (Appendix B.2). These design records link design abstractions to backend results, which supports studies such as the representative tasks in Section 4.

ID	Family	Input	Output	Allowed tools
T01	Implementation prediction	RTL, constraints	Routed LUT, FF, DSP, BRAM, frequency	`list_files`, `read_file`, `search`, `submit_answer`
T02		RTL, constraints, 10 candidate paths	Select the three most timing-critical paths	T01 tools
T03		HLS C, constraints	T01 output	T01 tools
T04		T03 input, generated RTL, HLS report	T01 output	T01 tools
T05	Behavioral prediction	RTL, waveform up to a chosen cycle, then clock and top-level inputs for 50 more	One selected signal's trace over the next 50 cycles	T01 tools, `list_signals`, `read_signals`
T06		HLS C, project configuration	Worst-case HLS synthesis latency	T01 tools
T07		T06 input, generated RTL, HLS reports, C testbench	Average co-simulation latency	T01 tools
T08	Ranking	10 RTL design variants of one base design	Relative synthesis area	T01 tools
T09		10 HLS design variants of one base design	Relative synthesis LUT, FF, DSP, BRAM, frequency	T01 tools
T10		T09 input, generated RTL, HLS reports	T09 output	T01 tools
T11	Generation	RTL with one segment removed, waveform	Missing RTL segment	T05 tools, `write_file`, `patch_file`, `rtl_check`, `rtl_sim`
T12		T11 input, HLS C	Missing RTL segment	T11 tools
T13		HLS C with one segment removed, generated RTL	Missing HLS segment	T01 tools, `write_file`, `patch_file`, `csim`, `csynth`

Table 3: Inputs, outputs, and tools of the 13 built-in tasks.

4 Custom Benchmarks

4.1 Construction Skill and Evaluation Harness

A benchmark specifies which design records to use, what information a model can access, and how to score submitted answers. We provide the `build-benchmark` agent skill to construct benchmarks from ChipFactory design records and `chipfactory-harness` to execute them under a common protocol. Researchers can adapt the built-in tasks (Section 4.2) or define custom tasks.

Construction skill. Researchers specify a design selection policy, task goal, and evaluation method. The skill resolves missing requirements and checks them against available data and tools. For example, post-route resource prediction requires design records with completed routing reports. It produces a manifest defining the selected problems, model-visible inputs, hidden references, permitted tools, budgets, and scoring procedures.

Evaluation harness. `chipfactory-harness` uses Pi (Zechner, 2025) to run each problem as an episode in a workspace containing model-visible artifacts. Models inspect designs with file and waveform tools; generation tasks also allow simulation and synthesis with the design record's testbench and flow. The harness enforces tool permissions and budgets and archives every request. The evaluator scores the final `submit_answer` against hidden references outside the workspace. Appendix C gives task construction and scoring details.

4.2 Task Suite

We define 13 built-in tasks in four families that researchers can use or adapt through `build-benchmark` (Table 3).

- **T01–T04: Implementation prediction** tests whether models can anticipate resource use and timing without running the backend. Scalar predictions in T01/03/04 are scored by the fraction of targets within 20% relative error (Appendix C.2). T02 asks models to select the three most timing-critical paths from ten candidates and awards credit only when all three are correct.

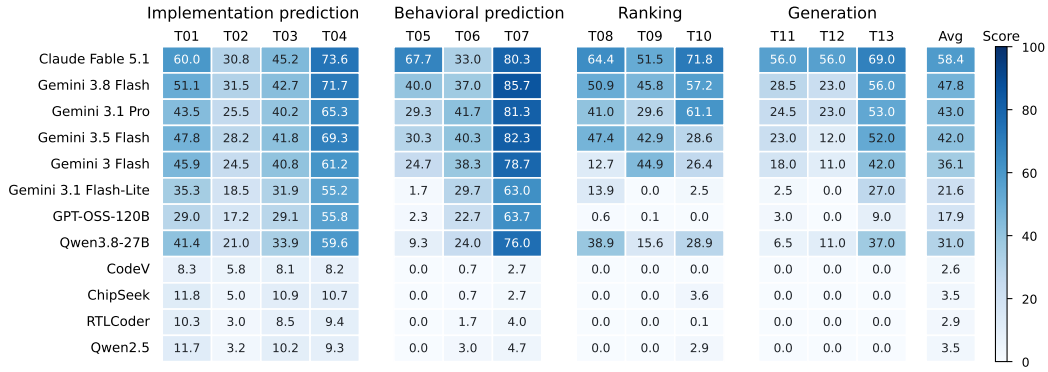


Figure 5: Task scores across four task families. Avg equally weights all 13 tasks.

- **T05–T07: Behavioral prediction** tests temporal reasoning through signal forecasting and latency prediction. T05 uses exact match over the complete 50-cycle future trace; partial traces receive no credit. T06 and T07 use the same 20% error threshold as the other numerical prediction tasks.
- **T08–T10: Ranking** tests whether models can compare different implementations of the same base design. We measure agreement between the predicted and reference orderings of all ten design variants using Spearman correlation.
- **T11–T13: Generation** tests whether models can reconstruct missing code within a larger design. T11 and T12 require compilation and whole-testbench success, with the final quarter of the testbench hidden. T13 requires hidden C-testbench success and HLS synthesis with estimates matching the reference within specified tolerances.

5 Experiments

5.1 Benchmark evaluation setup

Models and problems. We report results for twelve models: six closed models (Claude Fable 5.1 (Anthropic, 2026) and five Gemini variants (Google DeepMind, 2026)) and six open-weight models (Qwen2.5-Coder (Hui et al., 2024), RTLCoder (Liu et al., 2024a), ChipSeek (Chen et al., 2026), CodeV (Zhao et al., 2024), GPT-OSS-120B (OpenAI, 2025), and Qwen3.8-27B (Qwen Team, 2026)). All twelve models have results on all 13 tasks. Problems are sampled from ChipFactory using source-stratified random sampling subject to task eligibility: T01–T04 contain 400 problems each, T05–T07 contain 300 each, T08–T10 contain 80 each, T11 contains 200, and T12–T13 contain 100 each. Each ranking problem comprises ten design variants. T03/T04, T06/T07, and T09/T10 share design records, while T12 uses a subset of the records selected for T11. Appendix D.1 provides the selection rules and model configurations.

Execution and scoring. Each problem is evaluated in one agent episode, with baseline limits of 600 seconds and 20 turns for T01–T10, and 3,600 seconds and 40 turns for T11–T13; token budgets are specified in Appendix D.1 and tool permissions in Table 3. Scores range from 0 to 100: pass rates for pass/fail tasks and success rates within 20% relative error for numerical prediction. Ranking averages Spearman correlations equally across metrics, floors the final mean at zero, and multiplies by 100. Invalid or missing answers receive zero credit and remain in the denominator.

5.2 Benchmark results and analysis

Overall performance. Figure 5 summarizes task scores across the evaluated models. Claude leads T01, T03, and T04, scoring 60.0, 45.2, and 73.6, respectively. Gemini 3.1 Pro leads latency prediction on T06 with 41.7, while Gemini 3.8 Flash leads T07 with 85.7. On full 50-cycle trace prediction, Claude, Gemini 3.8 Flash, and GPT-OSS-120B score 67.7, 40.0, and 2.3, respectively.

Generation also separates the evaluated models. Claude passes 56%, 56%, and 69% of T11–T13 problems, compared with 3%, 0%, and 9% for GPT-OSS-120B. Qwen3.8-27B passes 6.5%, 11%, and 37% of T11–T13 problems, while the four 6.7–7B models score zero on all three tasks. Even the highest-scoring model therefore leaves a substantial fraction of reconstruction problems unsolved. Appendix D.2 provides valid-answer rates and analysis details.

Failure cases on Claude. Compilation and synthesis errors account for 100 of the 163 failed problems (Table 4). Simulation failures and failures in timing/resource checks follow, with 21 problems each. The remaining failures involve missing or invalid submissions (13) and verification timeouts (8).

Failure category	T11	T12	T13	Total
Missing or invalid submission	11	2	0	13
Compilation / synthesis error	57	39	4	100
Simulation failure	14	2	5	21
Timing/resource check failure	0	0	21	21
Verification timeout	6	1	1	8

Table 4: Claude fails 163/400 problems.

Cross-level context. Figure 6 compares four matched task pairs for all twelve models. Adding generated RTL and HLS reports improves T03→T04 scores for ten of the twelve models; Gemini 3.8 Flash rises from 42.7 to 71.7. The same additions have mixed effects on ranking: Gemini 3.1 Pro gains 31.4 points on T09→T10, while Gemini 3 Flash loses 18.5. Adding HLS source to reconstruction changes T11→T12 pass rates by −5 to +4 percentage points on their common 100-problem subset. The value of cross-level context therefore depends on the task and model.

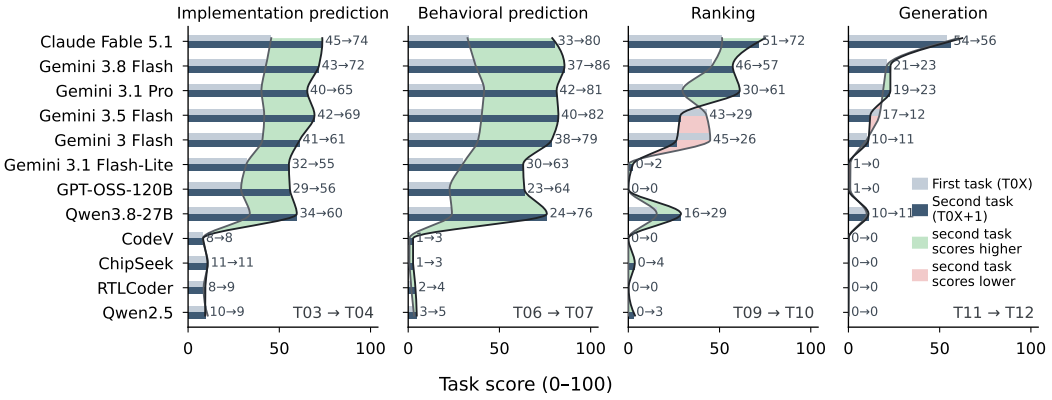


Figure 6: Scores on matched task pairs. Light and dark bars denote the first and second task, respectively. Curves follow their scores across model rows; green and red shading mark where the second task scores higher and lower.

Design complexity. Figure 7 examines one RTL task per family: implementation prediction (T01), signal forecasting (T05), area ranking (T08), and RTL reconstruction (T11). For each task, we divide problems into three groups by RTL size, with approximately equal numbers of problems in each group. T08 uses the mean RTL size over its ten design variants. All twelve models score lower in the largest T01 bin than in the smallest. Claude decreases from 67.8 to 50.0 on T01 and from 77.5 to 57.4 on T05. T08 and T11 show no consistent monotonic trend across models. For T08, ranking variants of the same base design may depend more on relative differences between variants than on total RTL size. For T11, the missing region contains fewer than 1,000 lines; once the design is sufficiently large, further increases in size may have little effect, as the additional RTL may be largely unrelated to the missing region.

5.3 Supervised prediction using ChipFactory

In addition to the 13 frozen-model tasks, ChipFactory supports training task-specific predictors of implementation outcomes. For RTL, we use MasterRTL (Fang et al., 2023) and RTL-Timer (Fang et al., 2024b) with their released configurations. For mapped netlists, we compare XGBoost (Chen & Guestrin, 2016) on scalar circuit features with DE-HNN (Luo et al., 2024) on the circuit hypergraph. We evaluate both held-out variants of seen base designs and entirely unseen base designs. These settings measure within-family generalization and transfer to new designs, respectively.

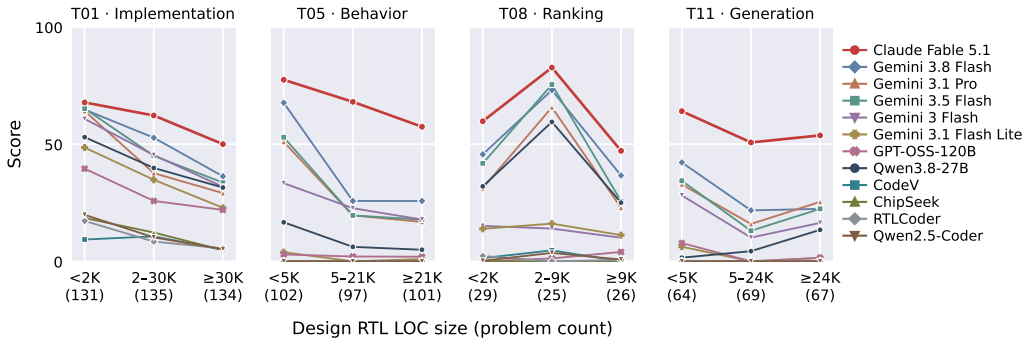


Figure 7: Task scores by complete reference RTL size. Parentheses indicate problem counts.

		Unseen base designs			Held-out design variants		
Input	Model	MAE↓	$\rho\uparrow$	$\tau\uparrow$	MAE↓	$\rho\uparrow$	$\tau\uparrow$
Netlist	XGBoost	48.8	0.940	0.807	30.6	0.972	0.868
Netlist	DE-HNN	43.7	0.961	0.855	27.0	0.979	0.874
RTL	MasterRTL	68.8	0.905	0.749	32.5	0.965	0.859
RTL	RTL-Timer	67.2	0.913	0.761	32.3	0.964	0.857

Table 5: Supervised maximum-frequency prediction on identical test sets. MAE in MHz; ρ/τ are Spearman/Kendall correlations.

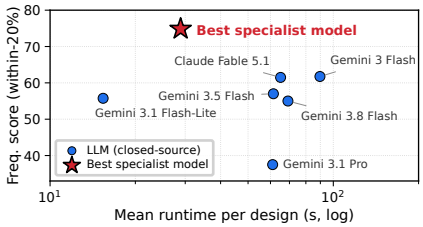


Figure 8: Accuracy versus end-to-end runtime on the T01 designs: six closed-source LLMs and the best specialist model.

Table 5 reports mean absolute error and rank correlations; Appendix D.3 gives the data splits and model settings. On unseen base designs, the models achieve rank correlations of $\rho = 0.905$–0.961. DE-HNN on mapped netlists gives the lowest MAE, 43.7 MHz, and the highest Spearman correlation, $\rho = 0.961$. Held-out variants are easier to rank ($\rho \geq 0.964$). Applied to the 400 designs of implementation-prediction task T01 (Section 5.2), the supervised predictors score higher than every frozen model at comparable end-to-end runtime (Figure 8), though they rely on Yosys (Wolf & Glaser, 2013) to construct their circuit representation and are trained on ChipFactory, whereas the language models operate on raw RTL with frozen parameters. Beyond evaluation, ChipFactory's backend feedback can thus supervise fast domain-specific predictors that agentic frameworks may invoke as tools when reasoning about implementation outcomes.

6 Conclusion

We present ChipFactory, a dataset covering complex hardware across diverse sources, abstraction levels, and implementation choices, with rich EDA backend feedback from simulation through physical implementation. Our evaluation of twelve models across 13 tasks reveals challenges in predicting implementation outcomes and reconstructing hardware, and shows how additional design context affects model performance. Reusable agent skills and an open evaluation harness allow researchers to extend these studies to new benchmarks, while the linked design representations and backend outcomes provide data for training specialized models. ChipFactory is open source to support community efforts to understand and improve AI capabilities for chip design.

Current LLMs leave a clear gap on every task in our benchmark. Claude Fable 5.1 is the strongest model, yet its average score is only 58.4 out of 100. The hardest task is critical-path selection (T02), where no model identifies the three most timing-critical paths in more than a third of the designs. The easiest task is co-simulation latency prediction (T07), where the best model still fails on one in seven designs. Open-weight models score at most 31.0 on average, which motivates using ChipFactory to further improve their chip design capability. The domain-specialized models score at most 3.5 on average and solve no generation problems. In contrast, supervised predictors trained on ChipFactory predict maximum frequency more accurately than every LLM, which shows the value of its backend labels.

AI USE STATEMENT

We used commercial and open-source AI models in this work. AI generated synthetic data for ChipFactory: LLMs write software specifications, and AI agents rewrite software programs and accelerators into HLS design variants (Section 3.2). Our experiments evaluate twelve AI models as benchmark subjects (Section 5). AI tools also helped us find related work, draft sections of this paper, and polish its writing. The authors reviewed all AI-assisted work and take full responsibility for the content of this paper.

ETHICS STATEMENT

All data in ChipFactory follow the licenses of their original sources. We are not aware of any other ethical issues with this work.

REPRODUCIBILITY STATEMENT

We will release all data, code, and agent trajectories upon acceptance.

REFERENCES

Stefan Abi-Karam and Cong Hao. HLS-Eval: A benchmark and framework for evaluating LLMs on high-level synthesis design tasks. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, pp. 219–226, 2025. doi: 10.1109/ICLAD65226.2025.00021.

Stefan Abi-Karam, Rishov Sarkar, Allison Seigler, Sean Lowe, Zhigang Wei, Hanqiu Chen, Nanditha Rao, Lizy John, Aman Arora, and Cong Hao. HLSFactory: A framework empowering high-level synthesis datasets for machine learning and beyond. In *2024 ACM/IEEE International Symposium on Machine Learning for CAD (MLCAD)*, 2024. doi: 10.1109/MLCAD62225.2024.10740213. URL https://arxiv.org/abs/2405.00820.

Tutu Ajayi, David Blaauw, Tuck-Boon Chan, C.-K. Cheng, Vidya A. Chhabria, D. K. Choo, M. Coltella, Ronald G. Dreslinski, Mateus Fogaça, S. Mehdi Hashemi, A. A. Ibrahim, Andrew B. Kahng, M. Kim, J. Li, Z. Liang, Uday Mallappa, Paul I. Pénzes, Geraldo Pradipta, Sherief Reda, Austin Rovinski, Kambiz Samadi, Sachin S. Sapatnekar, Lawrence K. Saul, Carl Sechen, Vaishnav Srinivas, William Swartz, Dennis Sylvester, D. Urquhart, L Wang, Mingyu Woo, and B. Xu. OpenROAD: Toward a self-driving, open-source digital layout implementation tool chain. In *Proceedings of the Government Microcircuit Applications and Critical Technology Conference (GOMACTECH)*, pp. 1105–1110, 2019. URL https://api.semanticscholar.org/CorpusID:210937106.

Christoph Albrecht. IWLS 2005 Benchmarks. International Workshop on Logic Synthesis (IWLS), June 2005. URL http://iwls.org/iwls2005/benchmarks.html.

Ahmed Allam and Mohamed Shalan. RTL-Repo: A benchmark for evaluating LLMs on large-scale RTL design projects. *arXiv preprint arXiv:2405.17378*, 2024.

Elisavet Lydia Alvanaki, Kevin Lee, and Luca P. Carloni. SLDB: An end-to-end heterogeneous system-on-chip benchmark suite for LLM-aided design. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, pp. 227–234, 2025. doi: 10.1109/ICLAD65226.2025.00016.

Anthropic. Introducing Claude Fable 5.1 and Claude Mythos 5.1. Anthropic, September 2026. URL https://www.anthropic.com/claude-fable-and-mythos-5-1.

Aman Arora, Andrew Boutros, Daniel Rauch, Aishwarya Rajen, Aatman Borda, Seyed Alireza Damghani, Samidh Mehta, Sangram Kate, Pragnesh Patel, Kenneth B. Kent, Vaughn Betz, and Lizy K. John. Koios: A deep learning benchmark suite for FPGA architecture and CAD research. *arXiv preprint arXiv:2106.07087*, 2021. URL https://arxiv.org/abs/2106.07087.

Yunsheng Bai, Atefeh Sohrabizadeh, Zongyue Qin, Ziniu Hu, Yizhou Sun, and Jason Cong. Towards a comprehensive benchmark for high-level synthesis targeted to FPGAs. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/8dfc3a2720a4112243a285b98e0d4415-Abstract-Datasets_and_Benchmarks.html. Datasets and Benchmarks Track.

Suhail Basalama and Jason Cong. Stream-HLS: Towards automatic dataflow acceleration. In *Proceedings of the 2025 ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA)*, 2025. doi: 10.1145/3706628.3708878. URL https://arxiv.org/abs/2501.09118.

Christopher Batten, Nathaniel Pinckney, Mingjie Liu, Haoxing Ren, and Brucek Khailany. PyHDL-Eval: An LLM evaluation framework for hardware design using python-embedded DSLs. In *2024 ACM/IEEE International Symposium on Machine Learning for CAD (MLCAD)*, 2024. URL https://github.com/cornell-brg/pyhdl-eval.

Jason Blocklove, Siddharth Garg, Ramesh Karri, and Hammond Pearce. Chip-Chat: Challenges and opportunities in conversational hardware design. In *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, pp. 1–6, 2023. doi: 10.1109/MLCAD58807.2023.10299874.

Zhuomin Chai, Yuxiang Zhao, Yibo Lin, Wei Liu, Runsheng Wang, and Ru Huang. CircuitNet: An open-source dataset for machine learning applications in electronic design automation (EDA). *Science China Information Sciences*, 2022. URL https://arxiv.org/abs/2208.01040.

Angshuman Chakravertty, Rahul Koshti, Buddhi Prakash Sharma, and Vinay Chamola. Benchmarking LLMs for verilog design flows, 2026. URL https://arxiv.org/abs/2607.22759.

Kaiyan Chang, Kun Wang, Nan Yang, Ying Wang, Dantong Jin, Wenlong Zhu, Zhirong Chen, Cangyuan Li, Hao Yan, Yunhao Zhou, Zhuoliang Zhao, Yuan Cheng, Yudong Pan, Yiqi Liu, Mengdi Wang, Shengwen Liang, Yinhe Han, Huawei Li, and Xiaowei Li. Data is all you need: Finetuning LLMs for chip design via an automated design-data augmentation framework. In *2024 61st ACM/IEEE Design Automation Conference (DAC)*, 2024. doi: 10.1145/3649329.3657356. URL https://doi.org/10.1145/3649329.3657356.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, 2016. doi: 10.1145/2939672.2939785.

Zhirong Chen, Kaiyan Chang, Zhuolin Li, Cangyuan Li, Xinyang He, Chujie Chen, Mengdi Wang, Haobo Xu, Yinhe Han, Huawei Li, and Ying Wang. ChipSeek: Optimizing verilog generation via EDA-integrated reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 25180–25201, 2026. doi: 10.18653/v1/2026.acl-long.1154. URL https://aclanthology.org/2026.acl-long.1154/.

Animesh Basak Chowdhury, Benjamin Tan, Ramesh Karri, and Siddharth Garg. OpenABC-D: A large-scale dataset for machine learning guided integrated circuit synthesis. *arXiv preprint arXiv:2110.11292*, 2021.

Luca Collini, Siddharth Garg, and Ramesh Karri. C2HLSC: Can LLMs bridge the software-to-hardware design gap? In *2024 IEEE LLM Aided Design Workshop (LAD)*, pp. 1–12, 2024. doi: 10.1109/LAD62341.2024.10691856.

Fan Cui, Chenyang Yin, Kexing Zhou, Youwei Xiao, Guangyu Sun, Qiang Xu, Qipeng Guo, Demin Song, Dahua Lin, Xingcheng Zhang, and Yun Liang. OriGen: Enhancing RTL code generation with code-to-code augmentation and self-reflection. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024. doi: 10.1145/3676536.3676830.

Fan Cui, Hongyuan Hou, Zizhang Luo, Chenyun Yin, and Yun Liang. HWE-Bench: Benchmarking LLM agents on real-world hardware bug repair tasks, 2026. URL https://arxiv.org/abs/2604.14709.

Matthew DeLorenzo, Vasudev Gohil, and Jeyavijayan Rajendran. CreativEval: Evaluating creativity of LLM-based hardware code generation, 2024. URL https://arxiv.org/abs/2404.08806.

Chenhui Deng, Zhongzhi Yu, Guan-Ting Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. ACE-RTL: When agentic context evolution meets RTL-specialized LLMs. *arXiv preprint arXiv:2602.10218*, 2026. URL https://arxiv.org/abs/2602.10218.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255. Ieee, 2009.

Zijian Ding, Yang Zou, Yizhou Sun, and Jason Cong. Can agents design better chips with a higher level abstraction? *arXiv preprint arXiv:2609.21157*, 2026a.

Zijian Ding, Yang Zou, Yizhou Sun, and Jason Cong. High level synthesis in the agentic era. *Integration*, pp. 102878, 2026b.

Cagri Eryilmaz. ChipCraftBrain: Validation-first RTL generation via multi-agent orchestration. *arXiv preprint arXiv:2604.19856*, 2026. URL https://arxiv.org/abs/2604.19856.

Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, and Zhiyao Xie. MasterRTL: A pre-synthesis PPA estimation framework for any RTL design. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023. doi: 10.1109/ICCAD57390.2023.10323951.

Wenji Fang, Mengming Li, Min Li, Zhiyuan Yan, Shang Liu, Hongce Zhang, and Zhiyao Xie. AssertLLM: Generating and evaluating hardware verification assertions from design specifications via multi-LLMs. In *2024 IEEE LLM Aided Design Workshop (LAD)*, 2024a. doi: 10.1109/LAD62341.2024.10691792.

Wenji Fang, Shang Liu, Hongce Zhang, and Zhiyao Xie. Annotating slack directly on your Verilog: Fine-grained RTL timing evaluation for early optimization. In *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*, 2024b. URL https://arxiv.org/abs/2405.04829.

Wenji Fang, Shang Liu, Jing Wang, and Zhiyao Xie. CircuitFusion: Multimodal circuit representation learning for agile chip design. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/7ffb43adf37b3eeaba559098bc084cc6-Abstract-Conference.html.

Jiahao Feng, Haiyan Qin, Zhiwei Xie, and Wang Kang. CircuitWeave: Topology-behavior alignment for executable multimodal RTL generation, 2026. URL https://arxiv.org/abs/2607.23523.

Aggelos Ferikoglou, Andreas Kakolyris, Dimosthenis Masouros, Dimitrios Soudris, and Sotirios Xydis. CollectiveHLS: A collaborative approach to high-level synthesis design optimization. *ACM Transactions on Reconfigurable Technology and Systems*, 18(1):1–32, 2024. doi: 10.1145/3702005.

Aggelos Ferikoglou, Despoina Tomkou, Dimosthenis Masouros, Dimitrios Soudris, and Sotirios Xydis. GNΩSIS: Lessons learned in generating a high-level synthesis dataset. *ACM Transactions on Architecture and Code Optimization*, 23(1):25:1–25:27, 2026. doi: 10.1145/3797035. URL https://doi.org/10.1145/3797035.

Lorenzo Ferretti, Jihye Kwon, Giovanni Ansaloni, Giuseppe Di Guglielmo, Luca Carloni, and Laura Pozzi. DB4HLS: A database of high-level synthesis design space explorations. *IEEE Embedded Systems Letters*, 13(4):194–197, 2021. doi: 10.1109/LES.2021.3066882.

Mingzhe Gao, Jieru Zhao, Zhe Lin, Wenchao Ding, Xiaofeng Hou, Yu Feng, Chao Li, and Minyi Guo. AutoVCoder: A systematic framework for automated verilog code generation using LLMs. In *2024 IEEE 42nd International Conference on Computer Design (ICCD)*, 2024. doi: 10.1109/ICCD63220.2024.00033.

Quentin Gautier, Alric Althoff, Pingfan Meng, and Ryan Kastner. Spector: An OpenCL FPGA benchmark suite. In *2016 International Conference on Field-Programmable Technology (FPT)*, pp. 141–148, 2016. doi: 10.1109/FPT.2016.7929519.

Razine Moundir Ghorab, Emanuele Parisi, Cristian Gutierrez, Miquel Alberti-Binimelis, Miquel Moreto, Dario Garcia-Gasulla, and Gokcen Kestor. NotSoTiny: A large, living benchmark for RTL code generation. *arXiv preprint arXiv:2512.20823*, 2025. URL https://arxiv.org/abs/2512.20823.

Benjamin Goldblatt, Paolo Pedroso, Farhad Modaresi, Ethan Sifferman, and Matthew R. Guthaus. HighTide: An agent-curated open-source VLSI benchmark suite. *arXiv preprint arXiv:2606.04126*, 2026. URL https://arxiv.org/abs/2606.04126.

Google DeepMind. Gemini model cards. Google DeepMind, 2026. URL https://deepmind.google/models/model-cards/. Includes the model cards used to document Gemini 3 Flash, Gemini 3.1 Pro, Gemini 3.1 Flash-Lite, Gemini 3.5 Flash, and Gemini 3.8 Flash.

Pingakshya Goswami, Masoud Shahshahani, and Dinesh Bhatia. MLSBench: A benchmark set for machine learning based FPGA HLS design flows. In *2022 IEEE 13th Latin America Symposium on Circuits and Systems (LASCAS)*, 2022. doi: 10.1109/LASCAS53948.2022.9789084.

Ce Guo and Tong Zhao. ResBench: A resource-aware benchmark for LLM-generated FPGA designs. In *Proceedings of the 15th International Symposium on Highly Efficient Accelerators and Reconfigurable Technologies*, pp. 25–34, 2025. URL https://doi.org/10.1145/3728179.3728192.

Benjamin Hawks, Jason Weitz, Dmitri Demler, Karla Tame-Narvaez, Dennis Plotnikov, Mohammad Mehdi Rahimifar, Hamza Ezzaoui Rahali, Audrey C. Therrien, Donovan Sproule, Elham E. Khoda, Keegan A. Smith, Russell Marroquin, Giuseppe Di Guglielmo, Nhan Tran, Javier Duarte, and Vladimir Loncar. wa-hls4ml: A benchmark and surrogate models for hls4ml resource and latency estimation. *ACM Transactions on Reconfigurable Technology and Systems*, 19(2):20:1–20:29, 2026. doi: 10.1145/3787490. URL https://doi.org/10.1145/3787490.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. Qwen2.5-Coder technical report, 2024. URL https://arxiv.org/abs/2409.12186.

Pengwei Jin, Di Huang, Chongxiao Li, Shuyao Cheng, Yang Zhao, Xinyao Zheng, Jiaguo Zhu, Shuyi Xing, Bohan Dou, Rui Zhang, Zidong Du, Qi Guo, and Xing Hu. RealBench: Benchmarking Verilog generation models with real-world IP designs. *arXiv preprint arXiv:2507.16200*, 2025.

M. Zafir Sadik Khan, Nowfel Mashnoor, Mohammad Akyash, Kimia Azar, and Hadi Kamali. SAGE-HLS: Syntax-aware AST-guided LLM for high-level synthesis code generation, 2025. URL https://arxiv.org/abs/2508.03558.

M. Zafir Sadik Khan, Kimia Azar, and Hadi Kamali. Bench4HLS: End-to-end evaluation of LLMs in high-level synthesis code generation, 2026. URL https://arxiv.org/abs/2601.19941.

Qiufeng Li, Rongqian Chen, Quan Cheng, Chengxuan Wang, Sizhe Tang, Chia-Tung Ho, David Z. Pan, Tian Lan, and Weidong Cao. PDAGENT-BENCH: Characterizing, grounding, and architecting LLM/VLM agents for VLSI physical design, 2026. URL https://arxiv.org/abs/2606.17253.

Zeju Li, Changran Xu, Zhengyuan Shi, Zedong Peng, Yi Liu, Yunhao Zhou, Lingfeng Zhou, Chengyu Ma, Jianyuan Zhong, Xi Wang, Jieru Zhao, Zhufei Chu, Xiaoyan Yang, and Qiang Xu. DeepCircuitX: A comprehensive repository-level dataset for RTL code understanding, generation, and PPA analysis. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, pp. 204–211, 2025. doi: 10.1109/ICLAD65226.2025.00029.

Zhe Lin, Zike Yuan, Jieru Zhao, Wei Zhang, Hui Wang, and Yonghong Tian. PowerGear: Early-stage power estimation in FPGA HLS via heterogeneous edge-centric GNNs. In *2022 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, pp. 1341–1346, 2022. doi: 10.23919/DATE54114.2022.9774682.

Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. Invited paper: VerilogEval: Evaluating large language models for verilog code generation. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023. URL https://arxiv.org/abs/2309.07544v2.

Shang Liu, Wenji Fang, Yao Lu, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. RTLCoder: Outperforming GPT-3.5 in design RTL generation with our open-source dataset and lightweight solution. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pp. 1–5. IEEE, 2024a. doi: 10.1109/LAD62341.2024.10691788. URL https://doi.org/10.1109/LAD62341.2024.10691788.

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. 60:1–60:9, 2024b. doi: 10.1145/3676536.3697118.

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, 2024. doi: 10.1109/ASP-DAC58780.2024.10473904.

Zhishang Luo, Truong Son Hy, Puoya Tabaghi, Michaël Defferrard, Elahe Rezaei, Ryan Carey, Rhett Davis, Rajeev Jain, and Yusu Wang. DE-HNN: An effective neural model for circuit netlist representation. In *Proceedings of the 27th International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2024. URL https://arxiv.org/abs/2404.00477.

Ruiyang Ma, Yunhao Zhou, Yipeng Wang, Yi Liu, Zhengyuan Shi, Ziyang Zheng, Kexin Chen, Zhiqiang He, Lingwei Yan, Gang Chen, Qiang Xu, and Guojie Luo. DynamicRTL: RTL representation learning for dynamic circuit behavior. *Proceedings of the AAAI Conference on Artificial Intelligence*, 40(1):836–843, 2026. doi: 10.1609/aaai.v40i1.37051. URL https://ojs.aaai.org/index.php/AAAI/article/view/37051.

Bardia Nadimi, Ghali Omar Boutaib, and Hao Zheng. PyraNet: A multi-layered hierarchical dataset for Verilog. *arXiv preprint arXiv:2412.06947*, 2024.

Maohua Nie, Jiang Zhu, Jingqun Zhang, Zhichen Zeng, Jiayi Wang, Sibo Zhang, Jialin Wang, and C.-J. Richard Shi. HierSVA: A data synthesis pipeline, dataset, and benchmark for LLM-driven hierarchical hardware formal verification. *arXiv preprint arXiv:2606.13706*, 2026.

OpenAI. gpt-oss-120b & gpt-oss-20b model card. OpenAI model card, August 2025. URL https://openai.com/index/gpt-oss-model-card/.

Jingyu Pan, Chen-Chia Chang, Zhiyao Xie, Yiran Chen, and Hai Helen Li. EDALearn: A comprehensive RTL-to-signoff EDA benchmark for democratized and reproducible ML for EDA research. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024. doi: 10.1145/3676536.3697116.

Zedong Peng, Zeju Li, Mingzhe Gao, Qiang Xu, Chen Zhang, and Jieru Zhao. ForgeHLS: A large-scale, open-source dataset for high-level synthesis. *arXiv preprint arXiv:2507.03255*, 2025. URL https://arxiv.org/abs/2507.03255.

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 preprint arXiv:2506.14074*, 2025.

Suresh Purini, Siddhant Garg, Mudit Gaur, Sankalp Bhat, Sohan Mupparapu, and Arun Ravindran. ArchXBench: A complex digital systems benchmark suite for LLM driven RTL synthesis, 2025. URL https://arxiv.org/abs/2508.06047.

Ruidi Qiu, Grace Li Zhang, Rolf Drechsler, Ulf Schlichtmann, and Bing Li. AutoBench: Automatic testbench generation and evaluation using LLMs for HDL design. In *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD*, 2024. doi: 10.1145/3670474.3685956.

Ruidi Qiu, Grace Li Zhang, Rolf Drechsler, Ulf Schlichtmann, and Bing Li. CorrectBench: Automatic testbench generation with functional self-correction using LLMs for HDL design. In *2025 Design, Automation and Test in Europe Conference (DATE)*, pp. 1–7, 2025. doi: 10.23919/DATE64628.2025.10992873.

Qwen Team. Qwen3.8-27B model card. Hugging Face model card, 2026. URL https://huggingface.co/Qwen/Qwen3.8-27B.

Farzaneh Rabiei Kashanaki, Mark Zakharov, and Jose Renau. HDLEval: Benchmarking LLMs for multiple HDLs. In *2024 IEEE LLM Aided Design Workshop (LAD)*, 2024. doi: 10.1109/LAD62341.2024.10691770.

Yijun Shen, Minghao Shao, Yichen Zhao, Zhuoyan Yu, Boyuan Chen, Yik-Cheung Tam, and Muhammad Shafique. VHDLSuite: Unified pipeline for LLM VHDL generation with data synthesis and evaluation, 2026. URL https://arxiv.org/abs/2606.13735.

Arnav Sheth, Ivaxi Sheth, and Mario Fritz. ProtocolLLM: RTL benchmark for systemverilog generation of communication protocols, 2025. URL https://arxiv.org/abs/2506.07945.

Atefeh Sohrabizadeh, Yunsheng Bai, Yizhou Sun, and Jason Cong. Automated accelerator optimization aided by graph neural networks. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, pp. 55–60, 2022. doi: 10.1145/3489517.3530409. Final publication corresponding to the GNN-DSE project. The arXiv preprint appeared in 2021 as arXiv:2111.08848 under a different title.

Kimia Tasnia, Alexander Garcia, Tasnuva Farheen, and Sazadur Rahman. VeriOpt: PPA-aware high-quality verilog generation via multi-role LLMs, 2025. URL https://arxiv.org/abs/2507.14776.

Shailja Thakur, Baleegh Ahmad, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Ramesh Karri, and Siddharth Garg. VeriGen: A large language model for verilog code generation. *ACM Transactions on Design Automation of Electronic Systems*, 29(3):46:1–46:31, 2024. doi: 10.1145/3643681.

Pei-Huan Tsai, Kuan-Lin Chiu, William Baisi, Pin-Yu Chen, and Luca P. Carloni. HSCO-Bench: An agent-driven end-to-end hardware-software co-design benchmark for systems-on-chip, 2026. URL https://arxiv.org/abs/2605.19399.

Yun-Da Tsai, Mingjie Liu, and Haoxing Ren. RTLFixer: Automatically fixing RTL syntax errors with large language models, 2023. URL https://arxiv.org/abs/2311.16543.

Jing Wang, Shang Liu, Wenji Fang, Yuchao Wu, Yugao Zhu, and Zhiyao Xie. RTL-BenchLS: A large-scale benchmark for RTL reasoning and generation with large language models. *arXiv preprint arXiv:2606.08976*, 2026a. URL https://arxiv.org/abs/2606.08976.

Jinghua Wang, Lily Jiaxin Wan, Sanjana Pingali, Scott Smith, Manvi Jha, Shalini Sivakumar, Xing Zhao, Kaiwen Cao, and Deming Chen. OpenRTLSet: A fully open-source dataset for large language model-based Verilog module design. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, pp. 212–218, 2025. doi: 10.1109/ICLAD65226.2025.00038. URL https://arxiv.org/abs/2606.10285.

Mingjun Wang, Yihan Wen, Yuntao Lu, Fengrui Liu, Yuxiang Zhao, Boyu Han, Jianan Mu, Yibo Lin, Runsheng Wang, Bei Yu, and Huawei Li. CircuitNet 3.0: A multi-modal dataset with task-oriented augmentation for AI-driven circuit design. In *International Conference on Learning Representations*, 2026b. URL https://proceedings.iclr.cc/paper_files/paper/2026/hash/2e0f5561c1553a97cee5fa64575358c9-Abstract-Conference.html.

Yiting Wang, Guoheng Sun, Wanghao Ye, Gang Qu, and Ang Li. VeriReason: Reinforcement learning with testbench feedback for reasoning-enhanced verilog generation. In *Proceedings of the Great Lakes Symposium on VLSI 2026*, pp. 948–953, 2026c. doi: 10.1145/3787109.3815310. URL https://doi.org/10.1145/3787109.3815310.

Zhihai Wang, Zijie Geng, Zhaojie Tu, Jie Wang, Yuxi Qian, Zhexuan Xu, Ziyan Liu, Siyuan Xu, Zhentao Tang, Shixiong Kai, Mingxuan Yuan, Jianye Hao, Bin Li, Yongdong Zhang, and Feng Wu. Benchmarking end-to-end performance of AI-based chip placement algorithms, 2024. URL https://arxiv.org/abs/2407.15026. Introduces the ChiPBench benchmark; corresponds to the paper's ChiPBench-D source label.

Andy Wanna, Hanqiu Chen, and Cong Hao. ForgeBench: A machine learning benchmark suite and auto-generation framework for next-generation HLS tools, 2025. URL https://arxiv.org/abs/2504.15185.

Anjiang Wei, Huanmi Tan, Tarun Suresh, Daniel Mendoza, Thiago S. F. X. Teixeira, Ke Wang, Caroline Trippel, and Alex Aiken. VeriCoder: Enhancing LLM-based RTL code generation through functional correctness validation, 2025. URL https://arxiv.org/abs/2504.15659.

Zhigang Wei, Aman Arora, Ruihao Li, and Lizy John. HLSDataset: Open-source dataset for ML-assisted FPGA design using high level synthesis. In *2023 IEEE 34th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2023. doi: 10.1109/ASAP57973.2023.00040. URL https://lca.ece.utexas.edu/pubs/ASAP_2023_HLSDataset.pdf.

Clifford Wolf and Johann Glaser. Yosys – a free Verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, 2013.

Nan Wu, Hang Yang, Yuan Xie, Pan Li, and Cong Hao. High-level synthesis performance prediction using GNNs: Benchmarking, modeling, and advancing. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, pp. 49–54, 2022. doi: 10.1145/3489517.3530408.

Cunxi Yu, Chenhui Deng, Nathaniel Pinckney, and Brucek Khailany. Agentic hardware design as repository-level code evolution. *arXiv preprint arXiv:2606.28279*, 2026a. URL https://arxiv.org/abs/2606.28279.

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, 2026b. URL https://arxiv.org/abs/2601.21448.

Mario Zechner. pi: AI agent toolkit: unified LLM API, agent loop, TUI, coding agent CLI. https://github.com/earendil-works/pi, 2025. Version 0.87.0, MIT license.

Yongan Zhang, Zhongzhi Yu, Yonggan Fu, Cheng Wan, and Yingyan (Celine) Lin. MG-Verilog: Multi-grained dataset towards enhanced LLM-assisted verilog generation. In *2024 IEEE LLM Aided Design Workshop (LAD)*, 2024. doi: 10.1109/LAD62341.2024.10691738.

Yang Zhao, Di Huang, Chongxiao Li, Pengwei Jin, Muxin Song, Yinan Xu, Ziyuan Nan, Mingju Gao, Tianyun Ma, Lei Qi, Yansong Pan, Zhenxing Zhang, Rui Zhang, Xishan Zhang, Zidong Du, Qi Guo, and Xing Hu. CodeV: Empowering LLMs for verilog generation through multi-level summarization, 2024. URL https://arxiv.org/abs/2407.10424.

Qingyun Zou, Nuo Chen, Yao Chen, Bingsheng He, and WengFei Wong. HLStrans: Dataset for c-to-hls hardware code synthesis, 2025. URL https://arxiv.org/abs/2507.04315.

Qingyun Zou, Jiahao Cui, Nuo Chen, Bingsheng He, and Weng-Fai Wong. MHRC-Bench: A multilingual hardware repository-level code completion benchmark. *arXiv preprint arXiv:2601.03708*, 2026. URL https://arxiv.org/abs/2601.03708.

A FULL COMPARISON WITH PREVIOUS WORK

We manually checked every comparison entry against the corresponding papers and publicly available artifacts. The comparison covers 69 previous works across design complexity, design diversity, and backend feedback availability.

A.1 DESIGN COMPLEXITY

We measure the publicly released RTL subset of each work. N counts measured units rather than the full reported dataset. Each unit combines its selected RTL files, excluding testbenches.

LOC counts nonblank lines after removing file-level headers, while retaining internal comments. Files and modules count selected RTL files and HDL module/entity definitions. Ports count port-list entries, with each vector counted once. Depth is the longest resolved instantiation chain, or 1 when no instance edge is resolved; unresolved library cells and cross-language edges do not extend it. These measurements use source-level parsing and do not require successful synthesis. Gate-level and mixed RTL/netlist measurements are excluded.

A.2 DESIGN DIVERSITY

Figure 10 gives the abstraction levels and target platforms of the surveyed works. A representation present somewhere in a dataset does not establish a paired representation of every design.

A.3 BACKEND AVAILABILITY

Figure 11 shows the backend feedback stages recorded for each surveyed work. *Full Available* means that the corresponding results are publicly available across the dataset's applicable designs. *Not Applicable* is reserved for an inapplicable stage, such as HLS synthesis for a dataset consisting only of RTL designs. *Partial Available & On Demand* covers results released for only a subset, including example designs, or obtainable by running a provided flow. An applicable stage with neither released results nor an on-demand flow is left unmarked in the comparison. The recorded-stage list in Table 1 can include stages with only partial release coverage.

	Measured on	RTL LOC			Files		Modules		Depth		Ports	
	N	Mean	P50	P90	P50	P90	P50	P90	P50	P90	P50	P90
VHDLSuite (2026)	206	41.9	31	79	1	1	1	1	1	1	4	8
OpenRTLSet (2026)	131,309	89.5	29	183	1	1	1	1	1	3	5	17
MHRC-Bench (2026)	482	146.8k	7.46k	262.0k	37	217.9	36	255.8	4	9	399.5	4.36k
HSCO-Bench* (2026)	10	456	456	456	1	1	1	1	1	1	35	35
HighTide (2026)	17	60.4k	20.1k	120.2k	5	366.8	59	518	5	14.8	1.21k	6.36k
Bench4HLS (2026)	—	—	—	—	—	—	—	—	—	—	—	—
ChipBench (2026)	45	54.5	42	96.2	1	1	1	2	1	2	6	16.6
PDAgent-Bench (2026)	*Not yet released*											
CircuitWeave (2026)	4,917	29.5	14	80	1	1	1	1	1	1	4	14
LLM Verilog Flows (2026)	20	18.4	13	42.7	1	1	1	1	1	1	4	5.1
HierSVA (2026)	342	498	84.5	1.35k	2	20.9	2	20.8	2	6	11	132.5
HWE-Bench (2026)	—	—	—	—	—	—	—	—	—	—	—	—
RTL-BenchLS (2026)	9,698	66.3	50	120	1	1	1	1	1	1	8	22
wa-hls4ml (2026)	638,154	46.3k	36.4k	61.2k	20	29	21	33	3	3	891	1.66k
CircuitNet 3.0 (2026)	5,097	90.5	50	199	1	1	1	1	1	1	8	21
HLStrans (2025)	—	—	—	—	—	—	—	—	—	—	—	—
VeriReason (2025)	1,892	16.5	15	25	1	1	1	1	1	1	4	7
VeriCoder (2025)	125,777	28.9	22	58	1	1	1	1	1	1	5	11
SLDB (2025)	10	1.48k	1.14k	3.04k	4.5	11.7	8	14.4	4	5.2	55	200.7
SAGE-HLS (2025)	15,870	162.4	74	396.2	1	1	1	2	1	1	8	30
ResBench (2025)	7,560	13.4	10	27	1	1	1	1	1	1	3	7
RealBench (2025)	64	2.25k	967	3.89k	5	14.4	4.5	25.7	3	6	38.5	546
HLS-Eval (2025)	—	—	—	—	—	—	—	—	—	—	—	—
ForgeHLS (2025)	429,279	4.67k	998	7.95k	3	12	3	14	2	3	84	316
DeepCircuitX (2025)	8,014	12.4k	68	1.76k	1	12	1	15	1	3	8	123
CVDP (2025)	678	14.1k	118	5.18k	1	28	1	30	1	4	8	271.3
CircuitFusion (2025)	1	675	675	675	1	1	1	1	1	1	16	16
ArchXBench (2025)	—	—	—	—	—	—	—	—	—	—	—	—
VeriOpt (2025)	13,347	65.8	56	128	1	1	1	2	1	2	5	12
DynamicRTL (2025)	2	134.5	134.5	216.5	2	2	2	2	1	1	16	20.8
NotSoTiny (2025)	296	573.2	215	1.20k	1	1	4	9	2	4	23	70.5
ProtocolLLM (2025)	221	142.5	130	258	1	1	1	1	1	1	10	25
ForgeBench (2025)	—	—	—	—	—	—	—	—	—	—	—	—
HDLEval (2024)	190	26.5	13	51.7	1	1	1	1	1	1	3	9
VeriGen (2024)	108,971	389.1	98	685	1	1	1	2	1	2	8	44
RTL-Repo (2024)	4,085	1.78k	477	2.90k	7	18	8	21	2	4	51	204
PyraNet (2024)	692,238	78.6	36	170	1	1	1	1	1	1	5	17
PyHDL-Eval (2024)	168	23.2	19	42.6	1	1	1	1	1	1	4	7
OriGen (2024)	222,075	32.2	25	65	1	1	1	1	1	1	5	12
MG-Verilog (2024)	11,144	32	25	67	1	1	1	1	1	1	6	15
HLSFactory (2024)	196	3.38k	2.53k	7.43k	7	9.5	7	10	3	4	104	180.5
CreativEval (2024)	119	18.5	14	36.8	1	1	1	1	1	1	4	7
CorrectBench (2024)	156	15.9	13	28	1	1	1	1	1	1	4	7
CodeV (2024)	165,340	37	27	80	1	1	1	1	1	1	5	13
ChipGPT-V (2024)	41	49.1	33	99	1	1	1	3	1	2	5	16
ChipGPT-FT (2024)	194,004	299.8	91	600.7	1	1	1	1	1	1	9	44
AutoVCoder (2024)	191,464	33.1	23	72	1	1	1	1	1	1	4	10
AutoBench (2024)	156	15.9	13	28	1	1	1	1	1	1	4	7
AssertLLM (2024)	20	6.39k	2.17k	11.0k	11.5	34.4	18.5	49.5	4	7.1	187.5	804.6
ChiPBench-D (2024)	—	—	—	—	—	—	—	—	—	—	—	—
CollectiveHLS (2024)	—	—	—	—	—	—	—	—	—	—	—	—
C2HLSC (2024)	30	443.7	433	602	1	1	3	3	3	3	20	22
RTLLM 2.0 (2024)	50	47.5	38	99.3	1	1	1	1.1	1	1.1	5	9
EDALearn (2024)	84	75.6k	1.34k	412.0k	1	15.1	2.5	506.1	2	15	46	15.3k
CircuitNet 2.0 (2024)	8	201.9k	51.2k	493.7k	248	482.3	226	1.05k	9.5	14	5.09k	28.2k
VerilogEval (2023)	156	15.9	13	28	1	1	1	1	1	1	4	7
RTLFixer (2023)	156	15.9	13	28	1	1	1	1	1	1	4	7
RTLCoder (2023)	26,532	40.5	30	82	1	1	1	3	1	2	6	14
HLSyn (2023)	—	—	—	—	—	—	—	—	—	—	—	—
HLSDataset (2023)	9,931	13.1k	7.91k	28.2k	6	8	9	11	3	3	120	223
Chip-Chat (2023)	9	109.2	40	190.2	1	2	1	2.2	1	1.6	5	15.6
PowerGear (2022)	2	1.59k	1.59k	1.69k	11.5	11.9	13	13.8	4	4	101.5	110.7
HLS-Perf-GNN (2022)	1	246	246	246	2	2	3	3	3	3	23	23
MLSBench (2022)	975	10.1k	6.42k	21.9k	11	20	16	31	4	6	170	534.6
GNN-DSE (2021)	—	—	—	—	—	—	—	—	—	—	—	—
DB4HLS (2021)	—	—	—	—	—	—	—	—	—	—	—	—
Koios (2021)	20	23.8k	7.09k	41.3k	1	1	32.5	146.4	6	9	722	5.67k
OpenABC-D (2021)	29	9.02k	2.21k	9.40k	5	16.2	11	55	3	5.2	117	697.4
Spector (2016)	—	—	—	—	—	—	—	—	—	—	—	—
★ **ChipFactory**	**1,006,725**	**36.1k**	**8.78k**	**102.1k**	**6**	**51**	**20**	**234**	**4**	**8**	**623**	**8.88k**

*Top module only.

Figure 9: RTL complexity of previous works and ChipFactory. P50 and P90 denote the median and 90th percentile within each measured subset; $k = 10^3$. Dashes indicate missing or non-comparable measurements.

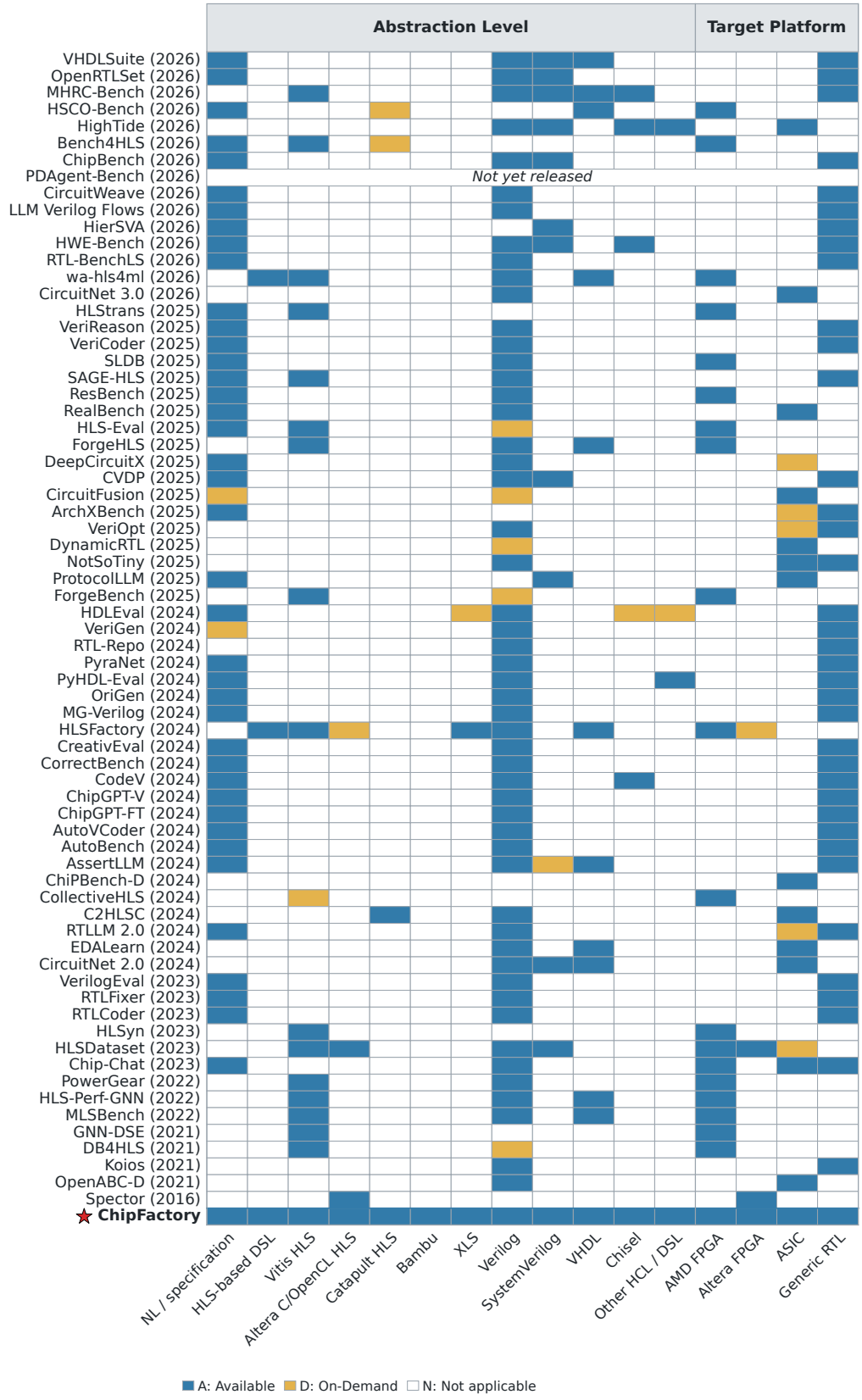


Figure 10: Abstraction levels and target platforms of the surveyed works.

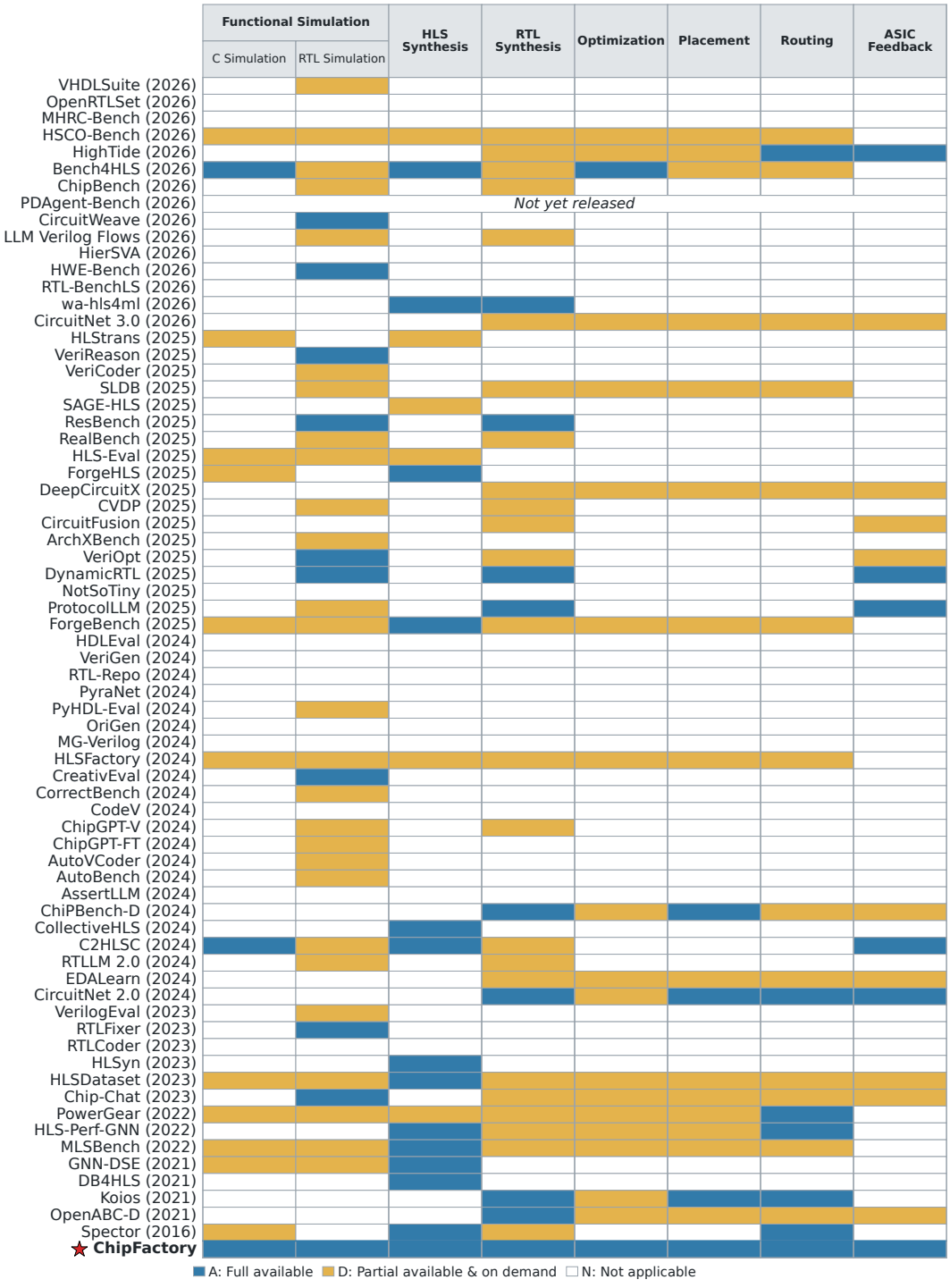


Figure 11: Backend feedback availability. Blue indicates available feedback; yellow indicates partial or on-demand feedback. Functional simulation is separated into C and RTL simulation.

B Dataset construction and coverage

B.1 Data preparation and expansion

When preparing a base design, we record where it comes from and how it is configured, reuse its tests when they exist, and use agents to write testbenches for software programs. For generators and compilers, we produce design variants by changing their input configurations, while for software programs and human-designed accelerators, we use agents to rewrite them into HLS design variants. HLS and HCL designs are compiled into RTL, whereas RTL designs from external datasets enter the backend directly. Throughout this process, we keep each original source apart from the designs derived from it, so that we can trace how each design variant was produced. We list the 38 external datasets and 32 generators and compilers used in ChipFactory in Table 6.

Source	Works
External datasets (38)	AutoBench (Qiu et al., 2024), Bench4HLS (Khan et al., 2026), C2HLSC (Collini et al., 2024), ChipBench (Yu et al., 2026b), ChiPBench-D (Wang et al., 2024), CircuitNet (Chai et al., 2022), CodeV (Zhao et al., 2024), CollectiveHLS (Ferikoglou et al., 2024), CorrectBench (Qiu et al., 2025), CreativEval (DeLorenzo et al., 2024), CVDP (Pinckney et al., 2025), ForgeBench (Wanna et al., 2025), ForgeHLS (Peng et al., 2025), GNN-DSE (Sohrabizadeh et al., 2022), GNΩSIS (Ferikoglou et al., 2026), HighTide (Goldblatt et al., 2026), HLSFactory (Abi-Karam et al., 2024), HLStrans (Zou et al., 2025), HLSyn (Bai et al., 2023), IWLS 2005 (Albrecht, 2005), Koios (Arora et al., 2021), MG-Verilog (Zhang et al., 2024), MLSBench (Goswami et al., 2022), OpenABC-D (Chowdhury et al., 2021), OpenROAD-flow-scripts (Ajayi et al., 2019), OpenRTLSet (Wang et al., 2025), OriGen (Cui et al., 2024), PowerGear (Lin et al., 2022), PyHDL-Eval (Batten et al., 2024), RealBench (Jin et al., 2025), RTL-Repo (Allam & Shalan, 2024), RTLFixer (Tsai et al., 2023), RTLLM 2.0 (Liu et al., 2024b), SAGE-HLS (Khan et al., 2025), SLDB (Alvanaki et al., 2025), VeriGen (Thakur et al., 2024), VerilogEval (Liu et al., 2023), and VeriOpt (Tasnia et al., 2025)
High-level compilers (17)	Allo, AutoSA, CODO, DaCe-FPGA, Dahlia/Fuse, FINN, fpgaConvNet-HLS, hls4ml, OPS-HLS, POM, Prometheus, PyLog, ScaleHLS/HIDA, SODA, ST-Flow, Stream-HLS, and Bambu
Chipyard generators (15)	AES, Ara, BOOM, Compress, CVA6, FFT, Gemmini, Ibex, Mempress, NVDLA, Radiance, Rocket, Sodor, Tacit, and VexiiRiscv

Table 6: External datasets, generators, and compilers in ChipFactory.

Taking the Stream-HLS `gemm` accelerator in Figure 3 as an example, we store its source, project configuration, testbench, and run script in `inputs/`, and the artifacts of each completed stage in `outputs/`. In `index.json`, we link the base design, the design variant, its source, and its target. In `status.json`, we record the execution plan, the outcome of each stage, and the run history.

For each design variant, we plan the last backend stage it should reach, such as RTL synthesis or routing, and run all earlier stages in the same design record. Tool versions, target devices or libraries, and constraints are recorded together with the results.

B.2 Detailed coverage and limitations

Design complexity. We apply the complexity metrics of Appendix A.1 to all 1,006,725 ChipFactory design records, grouping them by source class (Table 7). The median design has 8,779 lines of code, 20 modules, and a hierarchy depth of 4, and 44.8% of designs reach a depth of 5 or more. The software programs from our agent-driven scaling are both large and deep, with a median of 11,257 lines, and 51.8% of them reach a depth of 5 or more. In contrast, external datasets contribute the smallest and shallowest designs, with a median of 471 lines and 2 modules. Generators and compilers produce the deepest designs, and 74.2% of them reach a depth of 5 or more. By RTL origin, generated RTL is far deeper than hand-written RTL, with 41% of Vitis-generated, 86% of Bambu-generated, and 96% of SoC designs reaching a depth of 5 or more, compared with 1.0% of hand-written designs. For six HLS datasets that release only C/C++ (HLStrans, GNN-DSE,

Bench4HLS, ForgeBench, CollectiveHLS, and HLSyn), we are the first to measure the size and hierarchy depth of their RTL.

			LOC		Files		Modules		Ports		Depth		
Source	Designs	Bases	P50	P90	P50	P90	P50	P90	P50	P90	P50	P90	≥ 5
Generators and compilers	341,928	1,169	28,026	130,760	2	63	56	371	2,128	12,203	7	10	74.2%
External datasets	256,670	104,719	471	32,167	1	20	2	55	35	2,210	2	7	15.9%
Human-designed accelerators	247,654	5,938	7,490	96,336	9	85	9	140	426	6,503	4	7	29.8%
Software programs	160,473	14,429	11,257	39,816	14	38	22	74	722	2,580	5	7	51.8%
All	1,006,725	126,255	8,779	102,073	6	51	20	234	623	8,879	4	8	44.8%

Table 7: Design complexity by source class, as the median (P50) and 90th percentile (P90) per design and the share of designs with depth ≥ 5. The Bases column counts distinct base designs, and few bases with many designs indicate parameter sweeps of a few architectures.

Backend coverage. We report RTL origin, backend coverage, and construction cost in Table 8. The RTL-origin row covers the 1,006,725 design records, while the stage rows cover every design variant sent to the backend, including those that produced no RTL. Among the design records, 85,452 have a hierarchy depth of at least 2, keep both the C++ or HCL source and the RTL it produces, and complete placement or routing. The total of each stage also includes design variants that did not reach it, either because an earlier stage failed or because the run was not started. Waveforms are RTL-simulation records that keep full-signal traces, and construction compute is the sum of allocated vCPU-hours over all runs. Since different tools produce different artifacts, a design variant that completes a stage may not have every artifact listed in Table 2.

Count	Unit	Value
RTL origin: generated / hand-written / HCL-translated	design records	872,154 / 118,793 / 15,778
HLS synthesis	total / succeeded / failed	789,869 / 739,053 / 48,559
RTL synthesis	total / succeeded / failed	557,765 / 428,661 / 7,188
Placement	total / succeeded / failed	236,159 / 152,164 / 7,924
Routing	total / succeeded / failed	108,893 / 50,834 / 3,919
RTL simulation failed	design variants not implemented because RTL simulation failed	26,664
Waveforms	design records whose RTL simulation completed, each with a full-signal trace	86,337
Linked hierarchical designs	design records with depth ≥ 2, C++ or HCL source with its RTL, and placement or routing	85,452
Dataset size	bytes of the release	14.9 TB
Construction compute	allocated vCPU-hours over all runs that built the dataset	1.63M

Table 8: RTL origin, backend coverage, and construction cost. Failed runs are recorded as well, since they are useful training data.

Limitations. Our ASIC flows use open-source tools and PDKs (NanGate45, SkyWater 130, and ASAP7), and we acknowledge that their results may differ from commercial flows at advanced nodes. We leave calibrating and improving the open-source flows to future work. Not every design record includes a test, but these records remain useful for predicting resource use, timing, and latency, and for ranking design variants.

C BENCHMARK CONSTRUCTION AND EVALUATION

This appendix specifies the harness workflow and task rules for the 13-task suite in Section 4, followed by a generation example.

C.1 HARNESS WORKFLOW

The `build-benchmark` skill fixes each problem's inputs, hidden references, tools, budgets, and scoring procedure in a manifest. The harness loads the task prompt, answer schema, and visible files into a workspace. Figure 12 follows a T11 problem: the model receives masked RTL and a waveform prefix, while the complete reference and hidden tests remain outside its workspace. Its tool kit (Table 3) reads files and the waveform, edits the answer, and runs EDA tools that compile or simulate it. The EDA calls are optional, and the model can use their feedback to revise its answer. The first accepted `submit_answer` ends the episode; the evaluator checks code preservation, compilation, and the full testbench.

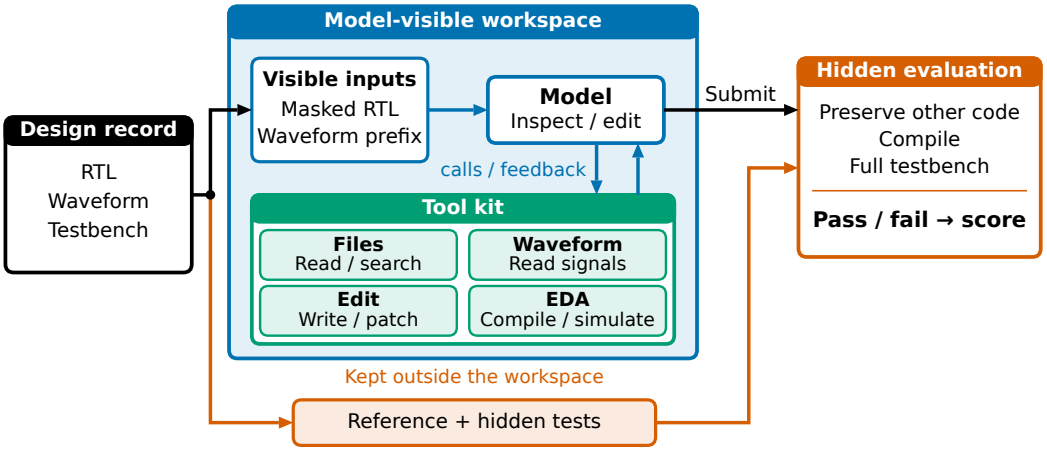


Figure 12: Harness workflow for RTL reconstruction (T11). The model works with masked RTL and a waveform prefix; references and hidden tests remain outside its workspace. The model calls tools from the tool kit and reads their feedback.

The harness reports remaining budgets and a final-turn notice. Each episode records the manifest digest, requests, tool responses, answer, usage, and outcome. Invalid or missing answers, context exhaustion, and timeouts contribute zero and remain in the applicable scoring denominator. Infrastructure failures follow the recorded retry policy; candidate compilation and test failures remain candidate outcomes. Prediction and generation task scores are 100 times the mean problem score, while ranking uses the aggregation below. The overall Avg equally weights all thirteen tasks; Valid Answer Rate separately measures the fraction of scoreable submissions. Problem counts, model configurations, budgets, and protocol exceptions are in Appendix D.1.

C.2 TASK SPECIFICATIONS

Each task specifies eligibility, visible inputs, problem construction, and scoring. Prediction and ranking inputs exclude target labels and files that directly report them. Numerical error uses

$$r(p, y) = \frac{|p - y|}{d(y)}, \qquad d(y) = \begin{cases} |y|, & y \neq 0, \\ 1, & y = 0. \end{cases}$$

For a zero reference, the denominator is one in the metric's native units.

T01. Routed resource and frequency prediction from RTL

Eligibility. FPGA RTL with routed utilization and timing reports, one constrained clock, and a nonempty intra-clock setup group.

Visible inputs. RTL and constraints.

Construction. Extract LUT, FF, DSP, BRAM in 18-Kbit equivalents, and frequency $1000/(P - W)$ MHz from the routed reports. Here P is the period and W the group's worst setup slack, both in ns; require $P - W > 0$ and include constrained I/O paths.

Test and score. Average five binary outcomes, each one when $r \leq 0.2$ and zero otherwise.

T02. Critical-path selection

Eligibility. FPGA or ASIC RTL with at least ten distinct setup paths in a routed timing report.

Visible inputs. RTL, constraints, and candidate paths with endpoints and cell sequences.

Construction. Sample ten paths from the report, shuffle their order, and remove delay and slack labels.

Test and score. One for a valid set of the three smallest-slack paths; otherwise zero. Include all paths below the third-place cutoff, fill remaining positions from ties, and ignore ID order.

T03. Routed resource and frequency prediction from HLS

Eligibility. FPGA HLS designs with project configuration and routed reports satisfying T01's timing requirements.

Visible inputs. HLS source and project configuration, including target and constraints.

Construction. Extract the five routed reference metrics as in T01; pair the problems with T04.

Test and score. The five-component rule in T01.

T04. Routed prediction with generated RTL

Eligibility. T03 designs with generated RTL and HLS synthesis reports.

Visible inputs. T03 inputs plus these artifacts.

Construction. Reuse T03 problems and references.

Test and score. T01 scoring.

T05. Signal forecasting

Eligibility. FPGA or ASIC RTL with a waveform and an internal signal changing at least six times in the prefix and twice in the next 50 cycles.

Visible inputs. RTL, all recorded signals for a 50–500-cycle prefix, and only clock and top-level inputs for the next 50 cycles.

Construction. Select the signal and window with a fixed seed; retain its future trace as the reference.

Test and score. One only for an exact match of the complete 50-cycle target trace.

T06. HLS synthesis latency prediction

Eligibility. FPGA HLS designs with a synthesis worst-case latency and a co-simulation average latency below 50,000 cycles.

Visible inputs. HLS source and project configuration.

Construction. Use the synthesis report's worst-case latency in cycles; T06 and T07 share designs.

Test and score. One when $r \leq 0.2$; otherwise zero.

T07. Co-simulation latency prediction

Eligibility. T06 designs with generated RTL, synthesis and scheduling reports, and a C testbench.

Visible inputs. T06 inputs plus these artifacts.

Construction. Use the recorded average co-simulation latency for the testbench workload, in cycles.

Test and score. The numerical rule in T06.

T08. Synthesis area ranking

Eligibility. Ten FPGA or ASIC implementations of one function, with varying measured areas and a common source family, tool build, target, and requested clock.

Visible inputs. The ten RTL implementations.

Construction. Group by specification and request nonnegative relative areas. FPGA area sums used/available fractions for LUT, FF, DSP, and BRAM; ASIC area counts standard cells, excluding macros and SRAM. Independently written implementations are not checked for functional equivalence.

Test and score. Spearman correlation uses average ranks for ties. For T08–T10, average problem correlations ρ_{ig} over the fixed applicable problems I_{tg} in each group g, then equally over groups G_t:

$$S_t = 100 \max\left(0, \frac{1}{|G_t|} \sum_{g \in G_t} \frac{1}{|I_{tg}|} \sum_{i \in I_{tg}} \rho_{ig}\right).$$

T08 has two groups: FPGA and ASIC area. Invalid answers and undefined correlations, including constant references or predictions, contribute zero; signed correlations are averaged before the final zero floor.

T09. Resource and frequency ranking from HLS

Eligibility. Ten Vitis HLS implementations of one kernel under a common backend context. Eligible metrics are fixed across models: require all ten labels, exclude all-zero metrics, and retain at least one varying metric.

Visible inputs. The ten HLS sources and target configurations.

Construction. Use synthesis LUT, FF, DSP, BRAM, and frequency as references, requesting nonnegative relative values.

Test and score. T08 aggregation with the five metrics as groups.

T10. Ranking with generated RTL

Eligibility. T09 groups with generated RTL and HLS synthesis reports.

Visible inputs. T09 inputs plus these artifacts for each implementation.

Construction. Reuse T09 groups, references, and eligible metrics.

Test and score. T09 scoring.

T11. RTL reconstruction

Eligibility. T05 designs with a removable RTL segment and a rerunnable simulation. The intact design must pass the evaluator and the empty segment must fail.

Visible inputs. Remaining RTL and the first three quarters of the module's active waveform, excluding signals inside the removed segment. Netlists and checkpoints are withheld.

Construction. Strip comments, then remove the largest eligible submodule of 200–1,000 lines, falling back to a contiguous fragment of the module body.

Test and score. Preserve code outside the segment, except for comments and whitespace. Compilation and full-testbench success, including the hidden final quarter, score one; otherwise zero. Full-testbench success can reflect functional agreement without cycle-by-cycle equivalence, as in Appendix C.3.

T12. RTL reconstruction with HLS source

Eligibility. T11 designs with Vitis HLS source, co-simulation, a removable C segment, and a passing original C testbench.

Visible inputs. T11 inputs plus HLS source.

Construction. Reuse the T11 RTL mask and waveform cutoff.

Test and score. T11 checks and scoring.

T13. HLS reconstruction

Eligibility. T12 designs that pass C and synthesis checks with intact source and fail with the selected segment empty.

Visible inputs. Remaining HLS source, project configuration, and generated RTL.

Construction. Strip comments and mask a function body or code segment, preferring the largest eligible function body of 50–400 lines.

Test and score. Require code preservation as in T11, hidden C-testbench success, and HLS synthesis. Every available resource, best/worst latency, initiation interval, and clock-period estimate must match within 5% relative error; zero references require exact zeros. One only if all checks pass.

C.3 Generation example

Problem. A T11 problem uses a Keccak coprocessor from the PandA-Bambu crypto designs, compiled to ASIC RTL by Bambu. The model receives a 6,657-line Verilog file with the 594-line body of module `PBI_kekka_coproc` removed and a waveform covering the first 6,828 of 9,105 cycles. In the visible waveform the address output `p_A_address0` changes 4,986 times, while `done_port` stays low. The missing segment instantiates the controller, the datapath, and two registers, and connects them through 167 wires.

Model replacement. One differing connection from the recorded Claude Fable 5.1 response is shown against the hidden reference:

Reference

```
controller_kekka_coproc Controller_i (
    .done_port(done_delayed_REG_signal_in),
```

Claude Fable 5.1

```
controller_kekka_coproc Controller_i (
    .done_port(done_port),
```

Test outcome. Over 12 turns, the model reads and searches the RTL, lists the waveform signals without reading their values, and calls `rtl_check` and `rtl_sim` once each. Its 549-line body omits the reference's one-cycle done register and 64-bit input register. The completion compiles and matches every compared output through cycle 6,828. The full co-simulation checks functional results against the C reference without requiring cycle-by-cycle equivalence to the original RTL. The completion passes and the problem scores one.

D EXPERIMENTAL DETAILS

D.1 BENCHMARK EVALUATION SETUP

Problem populations. All twelve models are evaluated on the 3,140 problems across 13 tasks, with one retained outcome per model and problem. T01–T04 contain 400 problems each, T05–T07 contain 300 each, T08–T10 contain 80 each, T11 contains 200, and T12–T13 contain 100 each. Each ranking problem comprises ten design variants. T02 and T05 each include 100 ASIC problems, T08 includes 20, and T11 includes 66; all other problems are drawn from FPGA designs. Task eligibility and construction follow Appendix C.2.

Episode budgets. Table 9 gives the baseline budgets. Within the tool-call limit, T11 and T12 permit 20 syntax checks and five simulation checks; T13 permits five C simulations and five HLS synthesis checks. Tool execution time is included in the episode's wall-clock budget. The harness can grant up to two submission-only requests after token, turn, or tool-call exhaustion; these requests do not extend the wall-clock limit.

Limit	T01–T10	T11–T13
Wall-clock time (s)	600	3,600
Model turns	20	40
Cumulative input tokens, cached included	300,000	500,000
Total output tokens, thinking included	64,000	100,000
Output tokens per turn	16,384	32,768
Tool calls	160	160

Table 9: Baseline episode limits. Model-specific exceptions are described below.

Model configurations. The harness requests `medium` thinking where supported. Temperature and seed use default values. For local inference, context limits are 262,144 tokens for Qwen3.8-27B, 32,768 for CodeV, ChipSeek, and Qwen2.5-Coder, and 16,384 for RTLCoder. Qwen3.8-27B uses a 1,200-second limit for T01–T10.

D.2 ADDITIONAL BENCHMARK RESULTS AND ANALYSIS

Valid answers. Table 10 reports Valid Answer Rate; task scores are already given in Figure 5. CodeV, ChipSeek, RTLCoder, and Qwen2.5-Coder produce no scoreable T11–T13 submissions; their zero reconstruction scores arise from submission failures.

	Implementation prediction				Behavioral prediction			Ranking			Generation			Avg
	T01	T02	T03	T04	T05	T06	T07	T08	T09	T10	T11	T12	T13	Rate (%)
Claude Fable 5.1	99.8	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	94.5	98.0	100.0	99.4
Gemini 3.8 Flash	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	81.5	78.0	95.0	96.5
Gemini 3.1 Pro	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0	96.2	100.0	70.5	66.0	96.0	94.5
Gemini 3.5 Flash	99.8	99.8	98.8	100.0	99.7	99.3	100.0	98.8	100.0	86.2	57.0	51.0	90.0	90.8
Gemini 3 Flash	100.0	99.2	100.0	100.0	97.3	100.0	100.0	97.5	96.2	75.0	72.5	81.0	89.0	92.9
Gemini 3.1 Flash-Lite	100.0	100.0	100.0	100.0	99.0	100.0	100.0	100.0	43.8	15.0	74.0	92.0	82.0	85.1
GPT-OSS-120B	96.2	100.0	97.8	96.5	99.0	94.3	97.0	85.0	85.0	58.8	31.5	39.0	65.0	80.4
Qwen3.8-27B	95.0	96.8	100.0	99.8	72.7	100.0	100.0	91.2	73.8	100.0	32.0	52.0	65.0	82.9
CodeV	100.0	100.0	100.0	100.0	85.0	100.0	100.0	73.8	53.8	63.8	0.0	0.0	0.0	67.4
ChipSeek	100.0	100.0	100.0	100.0	96.7	100.0	100.0	80.0	71.2	75.0	0.0	0.0	0.0	71.0
RTLCoder	100.0	97.0	99.5	100.0	82.3	100.0	100.0	58.8	11.2	10.0	0.0	0.0	0.0	58.4
Qwen2.5	100.0	100.0	100.0	100.0	99.3	100.0	100.0	90.0	51.2	56.2	0.0	0.0	0.0	69.0

Table 10: Valid Answer Rate as a percentage of planned episodes. Avg equally weights all thirteen task-level rates.

Numerical prediction errors. Figure 13 reports mean relative error for each numerical prediction target, using the error definition in Appendix C.2. Means include only valid answers; Table 10 reports their coverage. Errors span several orders of magnitude, and all values are retained without clipping.



Figure 13: Mean relative error (%) over valid numerical predictions; lower is better. Colors use a shared logarithmic scale. BRAM denotes 18K-block equivalents and Freq. denotes achieved maximum frequency. T06 predicts synthesis worst-case latency, while T07 predicts average co-simulation latency.

Ranking correlations. Figure 14 reports signed Spearman correlations for T08–T10. Invalid answers and undefined correlations contribute zero. Claude attains the highest mean correlation on all three ranking tasks.

D.3 Supervised post-route timing prediction

This section details the training protocol and data splits used in the supervised study of Section 5.3. The common dataset contains 11,686 FPGA designs with both RTL and mapped-netlist representations and completed routing. All models use identical test sets: 2,192 designs from unseen base designs and 1,695 held-out variants from seen base designs. XGBoost, MasterRTL, RTL-Timer, and DE-HNN are trained on the same corresponding training sets, with 8,761 designs for the unseen-base split and 5,099 designs for the variant split. All models directly predict achieved maximum frequency $F_{\max}$. Table 5 reports mean absolute error (MAE) in MHz, Spearman rank correlation ρ, and Kendall rank correlation τ. For the T01 comparison in Figure 8, the frequency-prediction score is the percentage of designs predicted within 20% relative error, using the numerical error definition in Appendix C.2. For mapped-netlist prediction, we report the best DE-HNN variant, HyperConv with virtual nodes, and use XGBoost (Chen & Guestrin, 2016) as a scalar-feature baseline. For

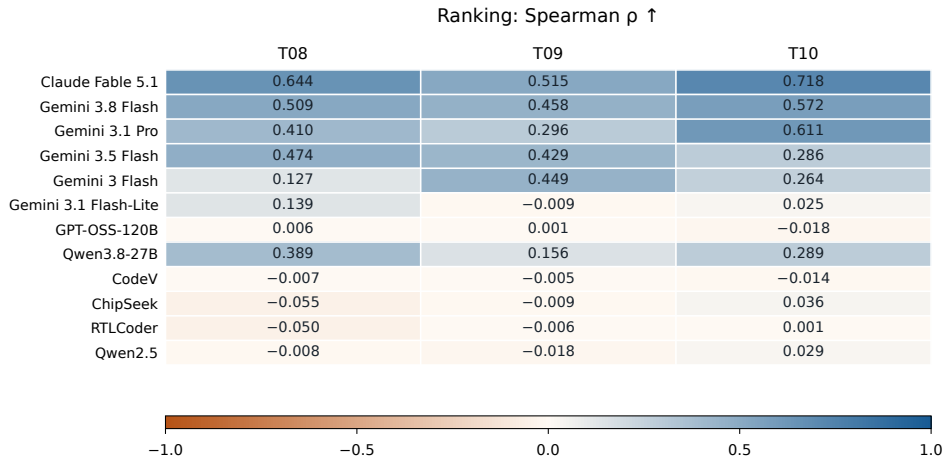


Figure 14: Signed ranking correlations for T08–T10. Negative correlations are retained; higher is better.

RTL prediction, MasterRTL and RTL-Timer operate on the same bit-level simple-operator-graph (SOG) representations using each repository's released XGBoost configuration. Both therefore use SOG structure, structural features, and design-level timing labels. We do not use open-source STA-derived timing annotations because the target FPGA architectures are not directly supported by the open-source EDA tools.

Test-set RTL sizes. RTL size counts nonblank, noncomment lines, with identical file contents counted once per design and testbench paths excluded. Table 11 gives the distribution over the two test sets and the common training set. Both test sets span the full complexity range, from a few hundred to over half a million lines (median 3.4K/5.9K lines, mean ≈25K), so the study is not restricted to small designs. Binned by the implementation-prediction cutpoints of Figure 7 (2K and 30K lines), the unseen-base test holds 927/848/417 small/medium/large designs and the variant test 637/561/497, giving substantial large-design (≥30K-line) coverage in both settings.

Split	n	Mean	p10	Median	p90	p99	<2K	2–30K	≥30K
Unseen base	2,192	25.7K	456	3.4K	62.4K	332K	927	848	417
Held-out variants	1,695	25.3K	678	5.9K	71.7K	183K	637	561	497
Common training set	8,761	22.8K	419	4.3K	54.4K	329K	3,548	3,215	1,998

Table 11: RTL size (nonblank, noncomment lines) of the supervised study's designs. The last three columns bin designs by Figure 7's implementation-prediction cutpoints (2K/30K lines).