WEP Warwick Evans Publishing

Transactions on Computer Science and Intelligent Systems Research
ISSN: 2960-1800, eISSN: 2960-2238 | Volume 13
AMMSAC 2026

A Verification-Guided Multi-Agent Framework for Reliable RTL Generation and Optimization

Haijian Zhang

Southeast University, Nanjing 210018, China

hj.zhang@ieee.org

Abstract. Large language models (LLMs) already emit syntactically plausible Verilog, yet a large share of generated designs fails functional verification, and the designs that do pass are not necessarily good implementations. We present VGMA, a verification-guided multi-agent framework that closes the loop between an RTL generator and open-source EDA tooling. VGMA synthesises a differential testbench directly from the reference port interface, compresses every failing simulation into a counterexample signature of nine semantic tags, maps that signature to a prior over repair operators, ranks suspicious lines with a backward cone-of-influence slice refined by the differing bit mask and by elaboration diagnostics, and admits a patch only after a bounded sequential equivalence proof. Every number reported here is produced by Icarus Verilog and Yosys. On 261 injected defects across 152 VerilogEval v2 problems, the full signal set places the faulty line first for 47.9% of defects (75.1% within the top five) against 31.4% (64.0%) without elaboration diagnostics, and repairs 73.3% of defects within 25 verification queries against 36.7% for an undirected search over an identical edit space, while using 40% fewer queries. On 220 real GPT-3.5-turbo and GPT-4 generations shipped with RTLLM, of which 125 fail, guided repair makes 40.0% of the 35 non-elaborating designs elaborate against 5.7% for undirected search, but end-to-end functional repair reaches only 5.6% against 1.6%: mutation corpora substantially overstate what line-local repair achieves on real LLM defects. Finally, 25.3% of functionally passing LLM designs synthesise larger than the human reference (mean 1.27x cells, worst case 18.8x), whereas a portfolio of synthesis scripts recovers only 2.0% on average, which places the optimisation opportunity at the RTL level rather than in the synthesis flow.

Keywords: Register-transfer level; Large language models; Multi-agent systems; Automated program repair; Fault localization; Functional verification; Equivalence checking; Logic synthesis.

1. Introduction

Register-transfer level (RTL) design is one of the most attractive targets for large language models (LLMs): the artefact is text, the specification is often already written in English, and the correctness criterion is machine-checkable by simulation. A sequence of benchmarks and models has appeared in rapid succession, from VerilogEval and its specification-to-RTL revision [1], [2] to RTLLM [3], VeriGen [4], [5], RTLCoder [6], CodeV [7] and agentic systems such as AutoChip [8], VerilogCoder [9], MAGE [10] and VFlow [11].

The headline metric of this literature is pass@k on a benchmark testbench, and it remains far from saturated on realistic designs. Re-running, under Icarus Verilog, the GPT-3.5-turbo and GPT-4 generations that ship with the RTLLM repository, we measure a functional pass rate of 52.7% for GPT-4 and 33.6% for GPT-3.5-turbo over 110 generations each (Table 3). Roughly one generated design in three is functionally wrong even for the stronger model, and one in ten does not elaborate at all. Single-shot generation is therefore not the interesting engineering problem; the interesting problem is what an agentic system does with a failure.

Existing closed-loop systems answer that question with a raw tool log. RTLFixer feeds compiler diagnostics back to the model through retrieval-augmented prompting and reports that about 55% of errors in LLM-generated Verilog are syntactic [12]; VerilogCoder adds abstract-syntax-tree-based waveform tracing [9]; MAGE decomposes the task across several LLM agents [10]. Three

weaknesses recur. First, the verification signal is consumed either as a pass/fail bit or as an unstructured text dump, so the agent must rediscover, in natural language, structure that the simulator already knows. Second, the oracle is whatever testbench the benchmark happens to provide, so a repair loop that is scored by that same testbench can silently overfit to it, an effect long documented in software repair [13], [14]. Third, functional correctness is treated as the terminal goal, although a design that passes may still be a poor implementation.

Adjacent domains show how much is gained by validating before committing. Zhao et al. put a pre-validation agent in front of deployment for cloud-native machine-learning pipelines, combining static dependency analysis with lightweight execution simulation to catch version conflicts and resource misconfigurations before submission, and raise the first-submission success rate from 57.2% to 82.2% while cutting resource over-provisioning by 31.2% [15]. Theirs is a compelling demonstration that the cheapest place to catch a generation error is before it reaches the target system. That is exactly the principle we specialise to RTL, with the advantage that a simulator and an equivalence checker make the pre-commitment check decisive rather than predictive.

This paper takes the position that verification, not generation, is the scarce resource in an agentic RTL flow, and that a framework should therefore (i) manufacture additional oracle strength cheaply, (ii) compress each failure into a machine-actionable form, (iii) spend its verification budget where the evidence points, and (iv) refuse to accept a patch that has not been proved equivalent to the reference. We instantiate this position as VGMA, a multi-agent framework built on Icarus Verilog [16] and Yosys [17], and we evaluate it on both a controlled defect corpus and a corpus of real LLM defects. Our contributions are:

• A unified verification signal that carries elaboration diagnostics, a differential simulation verdict, a counterexample signature and synthesis metrics through one interface, and an agent set (interface, syntax, verification, localisation, repair, equivalence, optimisation) that communicates only through it (Section III-A).

• Automatic differential-testbench synthesis from the reference port interface, including clock and reset inference. The synthesised testbench detects 242 of the 261 injected defects (92.7%) that the curated HDLBits testbenches detect, which makes a strong oracle available without human effort (Section III-B).

• A counterexample signature that reduces a failing trace to nine semantic tags, and a prior that maps tags to repair operators. This converts an undirected patch search into a guided one and improves top-1 fault localisation from 39.8% to 47.9% (Section III-C, III-D).

• An acceptance gate based on a bounded sequential equivalence proof through a Yosys SAT miter. Of the patches accepted by both testbenches, 81.8% are proved equivalent and none is refuted (Section V-E).

• An empirical result that we believe matters more than our own repair numbers: the same framework repairs 73.3% of single-site injected defects but only 5.6% of real LLM defects, whereas the elaboration-repair rate on real defects is 40.0%. Mutation corpora, which dominate the hardware-repair literature, therefore overstate end-to-end capability by more than an order of magnitude (Section V-D, VI).

• A measurement of implementation quality that motivates the optimisation agent: 25.3% of functionally passing LLM designs are larger after synthesis than the human reference, with a worst case of 18.8x, while a portfolio of five Yosys scripts recovers only 2.0% on average (Section V-F).

A scoping decision deserves to be stated immediately. The repair agent in this paper is deliberately instantiated with a deterministic operator search rather than an LLM. The framework is generator-agnostic and its interfaces accept an LLM backend, but keeping the backend deterministic isolates the contribution of the verification-guided orchestration from the stochasticity, the prompt sensitivity

and the unknown training data of a model, and it makes every number in Section V exactly reproducible from a fixed seed. Section VI discusses what this excludes.

2. Related Work

2.1. LLM-based RTL Generation

VerilogEval [1] established the now-standard evaluation protocol of 156 HDLBits problems with reference implementations and self-checking testbenches; its 2024 revision reframes the prompts as specification-to-RTL tasks and reports GPT-4o at 63% pass on that formulation [2]. RTLLM [3] moves towards design-level tasks, providing 50 designs with natural-language descriptions, golden RTL and testbenches, and ships the raw GPT-3.5-turbo and GPT-4 generations that we reuse as a defect corpus. Domain-adapted and fine-tuned models such as VeriGen [4], [5], RTLCoder [6], CodeV [7], OriGen [18] and HaVen [19] improve pass rates, and reinforcement learning against testbench feedback has been explored [20]. Recent work moves from single-shot prompting to agents: AutoChip iterates on compiler and simulator feedback [8], VerilogCoder plans over a task graph and traces waveforms through the AST [9], MAGE composes several specialised LLM agents [10], and VFlow searches over agentic workflows [11]. CVDP extends evaluation to a much broader set of design and verification tasks [21]. VGMA is complementary: it is the verification layer such systems need, and it is evaluated on the outputs of generators it does not control.

2.2. RTL Debugging And Repair

CirFix was the first automated program repair system for hardware description languages; it combines a wire- and register-assignment fault localisation with a genetic search over replace, insert and delete operators, and produces plausible repairs for 21 of 32 defect scenarios [22], carrying the search-based repair tradition established by GenProg [23] into hardware. Our undirected baseline is intentionally the same family of random operator search, applied to the same edit space as our guided agent, so that the comparison isolates guidance rather than operator design. RTLFixer targets elaboration errors with retrieval-augmented ReAct prompting and repairs 98.5% of compilation errors in a 212-design corpus derived from VerilogEval [12]; our elaboration experiment measures the same quantity on real RTLLM generations with a deterministic backend. LLM-based functional bug localisation in Verilog has also been studied directly [24], and assertion mining offers an orthogonal source of oracle strength [25].

2.3. Fault Localisation And Patch Overfitting

Spectrum-based fault localisation ranks program elements by the correlation between their execution and test failure, with Tarantula [26] and the Ochiai coefficient [27], [28] as canonical instances; a time-aware program spectrum has been adapted to hardware design code [29]. Our localisation agent is not spectrum-based: rather than executing many tests, it exploits the fact that a hardware failure exposes which output ports and which bit positions diverge, and slices backwards from exactly those. The risk that a patch satisfies the tests without being correct is well documented [13], [14], and generate-and-validate systems are known to produce plausible but incorrect patches [14]; we therefore make a formal equivalence check part of the acceptance criterion rather than a post-hoc audit.

Memory-augmented agents have recently pushed software localisation further. Mo et al. couple a code-specialised retrieval memory with a reinforcement-learned strategy memory and lift top-1 fault localisation on Defects4J from 42.6% to 47.6% while cutting editing churn by 31% [30]. We find their insistence on reporting debugging efficiency and run-to-run stability, rather than a single headline resolved-rate figure, a notably healthy way to evaluate a repair agent, and we adopt the same stance by reporting queries-to-fix beside repair rate throughout Section V.

2.4. Agentic Frameworks

The general agent literature supplies the control patterns we specialise: interleaving reasoning with tool calls [31], multi-agent conversation frameworks [32], and agent-computer interfaces for software engineering [33]. What distinguishes the hardware setting is that the tools are strong: a simulator and an equivalence checker can decide questions that in software require heuristics. VGMA is designed around that asymmetry.

Table 1. The nine counterexample-signature tag families and the repair operators they promote.

Tag	Trace condition	Promoted operators
stuck	candidate output constant, reference varies	sensitivity list, reset polarity, condition negation
inverted	candidate equals complement of reference	condition negation, unary not, ternary swap
lag1 / lead1	candidate equals reference shifted one cycle	assignment kind, clock-edge polarity
partial_bits	only some bit positions differ (mask kept)	bit-range, concatenation order, declaration width
const_offset	numeric difference is one constant	arithmetic operator, constant perturbation
reset_correlated	mismatch density in reset windows at least 2x	reset polarity, constant perturbation
burst	mean run of consecutive mismatches at least 2.5	FSM target, reset polarity, assignment kind
sporadic	isolated mismatches, mean run at most 1.3	relational, logical operator, ternary swap
sticky	mismatch never recovers before end of trace	reset polarity, FSM target, case default

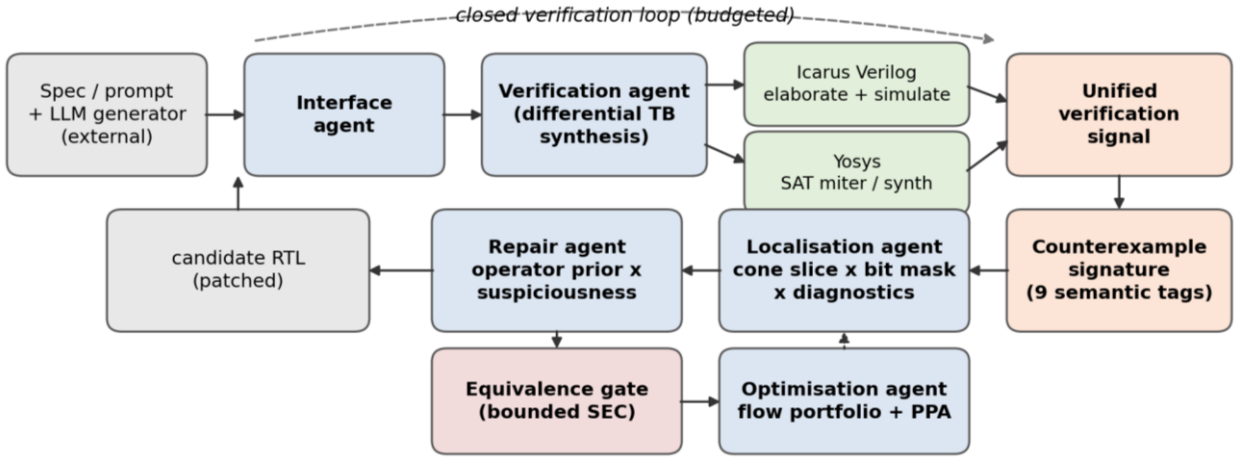

Figure 1. The VGMA agent set and the closed verification loop. Every agent exchanges a single unified verification-signal record; tool invocations (shaded, right) are the only source of measurement.

3. The Vgma Framework

3.1. Unified Verification Signal

Every agent in VGMA reads and writes one record, the verification signal, whose fields are: an elaboration flag; a list of (line, message) diagnostics together with a diagnostic class drawn from six patterns (undeclared identifier, port mismatch, syntax error, procedural assignment to a net, width

mismatch, unsupported construct); the verdict of the benchmark testbench; the verdict of the synthesised differential testbench; the counterexample signature described below; and, when requested, synthesis metrics and an equivalence verdict. Fig. 1 shows the resulting dataflow. The single-record discipline is what makes the ablations in Section V possible: a configuration is defined by which fields the repair agent is permitted to read, not by which code path runs.

Cost is accounted uniformly. One verification query is one elaborate-and-simulate cycle, regardless of which testbench it uses, and every configuration receives the same budget of 25 queries and the same wall-clock cap. Reported query counts include the initial diagnosis and the final confirmation.

3.2. Differential Testbench Synthesis

The verification agent parses the reference module interface, resolving both ANSI and non-ANSI port declarations, and classifies each input as a clock (by name), a reset (by name, with polarity inferred from an "_n" suffix) or a data input. It then emits a testbench that instantiates the candidate and the reference on identical stimulus, applies a deterministic reset pulse, drives data inputs with a seeded pseudo-random sequence at every active clock edge, re-asserts reset with low probability so that reset behaviour is exercised, and writes one trace line per sample containing the time, all data inputs and both output vectors. Comparison is against the reference rather than against expected values, so any stimulus is a sound differential test and no golden vectors are needed.

Two properties make this worth doing. It is free: the module interface is already available in any generation pipeline that knows what it asked for. And it is strong: on our injected corpus the synthesised testbench detects 92.7% of the defects that the hand-written HDLBits testbenches detect, and it additionally yields a per-sample trace, which the benchmark testbenches do not expose.

3.3. Counterexample Signature

A trace is far more informative than a mismatch count, but a repair agent cannot consume raw waveforms. The verification agent therefore reduces the trace, per output port, to a set of semantic tags (Table 1). Three families are useful in practice. Value-shape tags describe the algebraic relation between the candidate and reference outputs: stuck (the candidate output never changes while the reference does), inverted, const_offset (the numeric difference is a single constant), and partial_bits together with the exact differing bit mask. Timing tags describe the relation in time: lag1 and lead1 hold when the candidate reproduces the reference delayed or advanced by one cycle, which is the classic symptom of a blocking/non-blocking confusion or a misplaced register stage. Distribution tags describe when failures occur: burst (long runs of consecutive mismatching samples, indicating divergent sequential state), sporadic (isolated mismatches, indicating a combinational condition), sticky (the design never recovers), reset_correlated (mismatch density inside reset windows at least twice that outside), and input-gated tags naming an input that predicts failure.

Attributing a failure to particular steps by comparing aligned successful and failed executions has proved powerful in agent training. Rhee et al. introduce a step-level Failure Attribution Penalty that aligns a failed trajectory with a successful one by dynamic time warping and flags the steps whose action distributions diverge most in KL divergence; removing it costs 1.7 points of task success on GAIA [34]. It is an elegant answer to credit assignment when only an outcome label is available. Hardware verification is fortunate by comparison: the aligned successful execution comes for free from driving the reference model with identical stimulus, so the signature measures the divergence exactly instead of estimating it, which is what keeps the tag set of Table 1 cheap enough to recompute on every verification query.

3.4. From Signature To Repair Operators

The repair space is a library of 22 line-local edit operators covering assignment kind, reset and clock-edge polarity, logical and relational operator substitution, condition negation, constant and bit-range perturbation, shift direction, concatenation order, operand swap, ternary-branch swap, sensitivity-list

widening, declaration width, output-net promotion, statement deletion and finite-state-machine target retargeting. Enumerating every operator at every applicable site yields 42 candidate edits on average for designs of 17 non-blank lines, and several hundred for the larger ones, so the space is far too large to sweep inside a budget of 25 queries.

Guidance is the product of two independent factors. A tag prior maps each signature tag to the operators that plausibly explain it, taking the maximum weight over active tags; for instance reset_correlated raises reset-polarity edits, partial_bits raises bit-range and concatenation edits, lag1 raises assignment-kind edits, and a procedural-assignment diagnostic raises output-net promotion strongly. A suspiciousness score ranks source lines. The score of an edit is the product of the two, and the agent tries edits in descending order.

3.5. Fault Localisation

The localisation agent computes a backward cone of influence from exactly the output ports that the signature marks as failing, using a lightweight signal-dependency graph extracted from the candidate source, and adds three refinements: lines whose bit-select overlaps the differing bit mask are boosted; lines that match a syntactic pattern associated with an active tag are boosted (a reset identifier for reset_correlated, an assignment operator for lag1, a bracket for partial_bits); and lines named by an elaboration diagnostic dominate everything else. Table 4 reports the accuracy of the resulting ranking against known fault sites, and its degradation when the signature or the diagnostics are withheld.

3.6. Budgeted Repair Search And Acceptance

The repair agent first asks the interface agent to reconcile structural non-conformance, which for real generations means selecting the module whose port set best matches the reference and renaming it. It then evaluates the candidate, ranks the edit space, and tries edits one at a time, each costing one query. A candidate that satisfies the differential testbench is confirmed against the benchmark testbench before being reported, so a patch must satisfy two independent oracles. If no single edit succeeds and budget remains, the agent seeds a second pass from the highest-ranked single edits and from any edit that first made the design elaborate, and searches for two-edit repairs.

Finally the equivalence agent constructs a Yosys miter between the patched design and the reference, flattens it, and asks the built-in SAT solver to prove the equivalence assertion over a bounded number of cycles from the all-zero initial state. A patch is accepted only if the proof succeeds; a refutation identifies a testbench-overfitted patch, and a solver timeout is reported as inconclusive rather than as success.

3.7. Optimisation Agent

Once a design is correct, the optimisation agent measures cell count after generic gate mapping with ABC [36] and logic depth as the longest topological path, explores a portfolio of five Yosys scripts (the default flow, an area-directed ABC script, aggressive resource sharing, a re-synthesis script, and one-hot finite-state-machine re-encoding), keeps the best area and the best depth, and submits the chosen netlist to the equivalence gate before reporting any improvement. Improvements that the gate cannot confirm are reported as unaccepted.

Where such an agent should push is informed by learning-based physical design. Zhang et al. decompose multi-objective layout optimisation into a hierarchical policy pair with congestion-aware reward shaping and report 23.7% lower routing congestion, 18.4% better power efficiency and 15.2% better timing closure on OpenROAD benchmarks, with the margin over flat reinforcement learning widening as designs grow [35]. Their result that structured decisions taken early dominate the final quality of result is a useful guide for an agent operating one level higher: it suggests the leverage lies in the structure a generator commits to rather than in the script that maps it, and Section V-F reports exactly that asymmetry on our corpus.

Table 2. Controlled defect corpus (Corpus B) injected into VerilogEval v2 references.

Property	Value
Problems with verifiable reference	152 of 156
Injected detectable defects	261
Distinct problems covered	152
Injection operator classes	16
Elaboration / functional / hang	75 / 186 / 0
Detected by synthesised differential TB	242 (92.7%)
Mean candidate edits per design	42
Mean non-blank lines per design	17

4. Experimental Setup

4.1. Tools and Host

All measurements come from Icarus Verilog 12.0 (elaboration and simulation) and Yosys 0.33 (equivalence checking, synthesis statistics, longest topological path). The host is a single-core container, which constrains total experiment time and is the reason a small number of slow designs are excluded where stated. No commercial tool, liberty file or LLM inference is used anywhere in the measurement pipeline.

4.2. Corpus A: Real LLM Defects

The RTLLM repository [3] ships, alongside 50 designs with golden RTL and testbenches, the Verilog produced by GPT-3.5-turbo and GPT-4 over five independent trials each. Of the 29 designs for which generations exist, 22 have a golden reference that passes its own testbench under Icarus Verilog; the remaining seven use constructs our open-source flow cannot elaborate and are excluded. This yields 220 real generations, of which 125 fail (Table 3). We use no synthetic mutation in this corpus, and the failure modes are whatever the two models actually produced.

4.3. Corpus B: Controlled Defects

For questions that require a known fault site we inject single-site defects into the VerilogEval v2 references [2]. Of the 156 problems, 152 have a reference that passes both its shipped testbench and our synthesised differential testbench. We enumerate 16 injection operators chosen to mirror the error classes observed in Corpus A, sample them with a fixed seed while forcing distinct operators and distinct lines within a problem, and keep only defects that the shipped testbench actually detects, giving 261 defects over 152 problems. Omission defects (a deleted statement, a missing default branch) are not injected, because repairing them requires statement synthesis, which a line-local space cannot express; such defects do occur in Corpus A and are counted there as failures.

4.4. Configurations

Five configurations share one oracle, one edit space and one budget, and differ only in which fields of the verification signal the repair agent may read. A0 shuffles the edit space at random, which is the CirFix-style undirected search. A1 may read elaboration diagnostics only. A2 adds cone localisation from the failing output ports. A3 adds the counterexample-signature prior but no localisation. A5 is full VGMA. A fourth variant, A4, equals A5 with two-edit repair disabled and is discussed in Section V-C.

4.5. Metrics

Repair rate is the fraction of defects for which a patch satisfies both testbenches within budget. Queries-to-fix counts verification queries including diagnosis and confirmation. Localisation accuracy is top-k over the ranked line list, against the injected site. Equivalence outcome is one of

proved, refuted or inconclusive. Area is post-mapping cell count and depth is the longest topological path excluding flip-flops. All randomness is seeded; the raw measurement files and the scripts that aggregate them are released with the paper.

Table 3. Verification outcome of 220 real LLM generations for 22 RTLLM designs, measured with Icarus Verilog (5 trials per design per model).

Outcome	GPT-3.5	%	GPT-4	%	All
Pass	37	33.6	58	52.7	95
Functional fail	45	40.9	37	33.6	82
Elaboration fail	25	22.7	10	9.1	35
Timeout / hang	3	2.7	5	4.5	8
Total	110	100	110	100	220

Seven of the 29 designs with generations are excluded because their golden reference does not pass its own testbench under Icarus Verilog.

5. Results

5.1. How LLM-Generated RTL Fails

Table 3 quantifies the corpus. GPT-4 passes 52.7% of the 110 generations and GPT-3.5-turbo 33.6%, in line with the rates reported for these models on RTLLM [3], which validates our harness. The composition of the failures matters more than their number. Functional failures dominate for both models (37 and 45 cases), and elaboration failures are 9.1% for GPT-4 against 22.7% for GPT-3.5-turbo: capability improvement shows up mainly as fewer syntactic errors, not as proportionally fewer functional ones. Among the 35 elaboration failures the largest identifiable class is a procedural assignment to a net, that is, an output declared without reg and then assigned inside an always block. This single pattern accounts for the majority of cases whose message we can classify, and it is repaired by exactly one operator, which is why the elaboration results in Section V-D are so much better than the functional ones.

Eight generations neither fail nor pass but hang, exhausting the simulation timeout. These are typically combinational loops or clock-edge mistakes. A framework that treats simulation as a black-box pass/fail bit cannot distinguish them from functional failures, whereas the signature marks them with a timeout tag that raises clock-edge and assignment-kind operators.

Table 4. Fault-localisation accuracy on 261 injected defects (15.9 ranked candidate lines on average).

Signals used	top-1	top-3	top-5	mean rank
Signature + diagnostics + cone	47.9%	61.7%	75.1%	4.34
without CE signature	39.8%	57.5%	70.5%	4.83
without diagnostics	31.4%	47.5%	64.0%	5.63

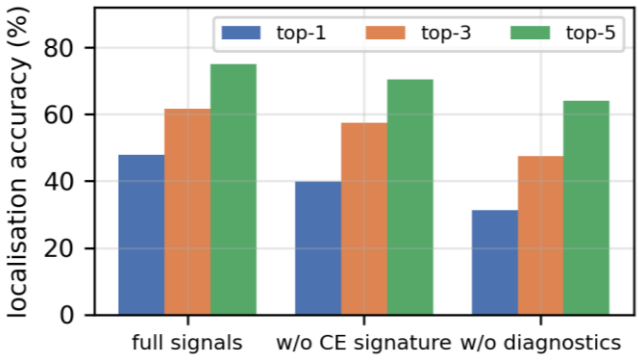

Figure 2. Fault-localisation accuracy on 261 injected defects when the counterexample signature or the elaboration diagnostics are withheld.

5.2. Localisation Accuracy

Table 4 and Fig. 2 report localisation over the 261 injected defects, where the ranked list contains 15.9 candidate lines on average. With the full signal set the faulty line is ranked first for 47.9% of defects and within the top five for 75.1%, at a mean rank of 4.34. Withholding the counterexample signature costs 8.1 points of top-1 accuracy, and withholding elaboration diagnostics costs 16.5 points. Both signals therefore contribute, and they contribute differently: diagnostics are decisive but only exist for elaboration failures, whereas the signature applies to every functional failure. This is the first quantitative justification for the signature as a component rather than as a heuristic.

Table 5. Repair ablation on 90 injected defects under an identical budget of 25 verification queries, one oracle and one edit space.

Configuration	n	fixed	rate	mean q	proved
A0 undirected search	90	33	36.7%	13.8	27/33
A1 diagnostics only	90	8	8.9%	18.6	-
A2 + cone localisation	90	58	64.4%	8.8	-
A3 + CE signature only	90	25	27.8%	11.6	-
A4 signature x cone	90	66	73.3%	8.3	-
A5 VGMA (full)	90	66	73.3%	8.3	54/66

"Proved" counts accepted patches for which bounded sequential equivalence to the reference was proved by the Yosys SAT miter; no accepted patch was refuted. The gate was run for A0 and A5 only.

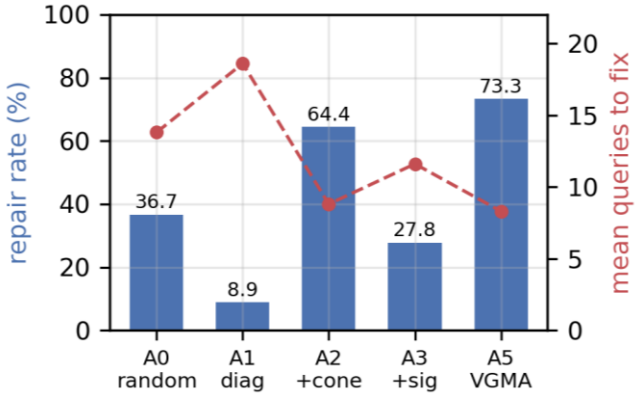

Figure 3. Repair rate (bars, left axis) and mean verification queries per successful repair (line, right axis) for the configurations of Table 5.

5.3. Repair Under A Verification Budget

Table 5 is the central ablation. Undirected search over the edit space repairs 36.7% of defects within 25 queries. Full VGMA repairs 73.3%, a factor of 2.00, and needs 8.3 queries per success against 13.8. Fig. 3 and Fig. 4 show that the advantage is not an artefact of the budget: the guided configuration is ahead at every budget from four queries upward, and its cumulative curve saturates while the undirected curve is still climbing.

Two ablation results are more interesting than the headline. First, guidance can be worse than no guidance. A1, which reads only elaboration diagnostics, repairs 8.9%, well below random, and A3, which reads only the signature prior, repairs 27.8%. The reason is structural: without a spatial ranking, an operator prior imposes a fixed sweep that applies the same high-prior operator to every line in file order, so it revisits irrelevant regions systematically, whereas random sampling at least spreads its budget. Verification feedback is not automatically useful; it becomes useful when it localises. Second, the two signals are complementary rather than redundant. Cone localisation alone (A2) reaches 64.4%, and adding the signature prior lifts it to 73.3% while reducing queries from 8.8 to 8.3.

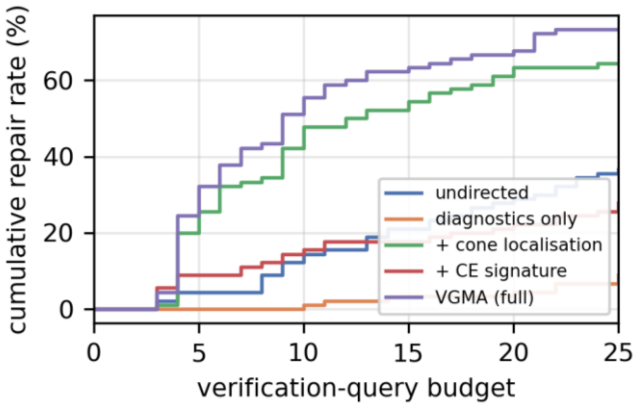

Figure 4. Cumulative repair rate versus verification-query budget. Guided configurations dominate at every budget above four queries.

Table 6. Repair rate of full VGMA by injected defect class (Corpus B).

Defect operator	n	fixed	rate
const pm1	8	2	25.0%
arith op	8	7	87.5%
assign kind	8	8	100.0%
logic op	8	6	75.0%
ternary swap	7	6	85.7%
concat order	7	7	100.0%
out net demote	7	7	100.0%
negate cond	7	5	71.4%
relational	7	7	100.0%
state target	7	2	28.6%
sens list	6	1	16.7%
range off1	4	4	100.0%
edge polarity	3	1	33.3%
reset polarity	2	2	100.0%
swap operands	1	1	100.0%

Two-edit repair contributes nothing on this corpus: configuration A4, identical to A5 but restricted to a single edit, also repairs 73.3% with 8.3 queries. This is expected, since every injected defect is

single-site, and it is precisely why a mutation corpus is insufficient on its own; the multi-edit pass earns its cost only on Corpus A. Table 6 breaks the full configuration down by defect class and shows where the operator space is adequate and where it is not.

5.4. Real Defects: A Large Validity Gap

Table 7 applies the same framework to the 125 real failing generations. Two very different pictures emerge depending on which sub-problem is asked.

Elaboration repair works. Of the 35 generations that do not elaborate, diagnostic-guided repair makes 14 (40.0%) elaborate within 25 queries, at 5.2 queries per success, against 2 (5.7%) for undirected search over the same space: a factor of 7.0. The gain comes almost entirely from mapping the diagnostic class to the right operator, which supports the design choice of classifying diagnostics rather than forwarding raw text.

End-to-end functional repair largely does not. Full VGMA repairs 7 of 125 real defects (5.6%) against 2 (1.6%) for undirected search; the relative advantage of guidance survives, at a factor of 3.5, but the absolute rate collapses from 73.3% on injected defects to 5.6%, and the mean query count rises to 22.9 of a budget of 25, meaning the search almost always exhausts its budget. The differential testbench was successfully synthesised for all 125 cases, so the loss is not a missing oracle. It is the patch space: a wrong LLM design is usually not a correct design with one wrong token, but a differently structured design, and no sequence of one or two line-local edits expresses the necessary rewrite. Fig. 5 contrasts the two regimes.

We regard this as the most actionable finding in the paper. The hardware-repair literature reports on curated or mutated defect suites [22], and our own controlled numbers would support an encouraging narrative. Measured on defects that a real model really produced, the same framework with the same verification signal is an order of magnitude weaker. The bottleneck is the expressiveness of the patch generator, not the quality of the feedback, which is a direct argument for placing a generative model inside a verification-guided loop of this kind rather than for improving the search.

Table 7. Corpus A: repair of real GPT-3.5-turbo and GPT-4 defects.

Task and method	n	fixed	rate	mean q
Elaboration, undirected	35	2	5.7%	5.5
Elaboration, guided	35	14	40.0%	5.2
Functional, undirected	125	2	1.6%	23.6
Functional, full VGMA	125	7	5.6%	22.9

Budget 25 queries in all four rows. The functional rows include the 35 non-elaborating designs, for which an elaboration fix alone is not counted as a repair.

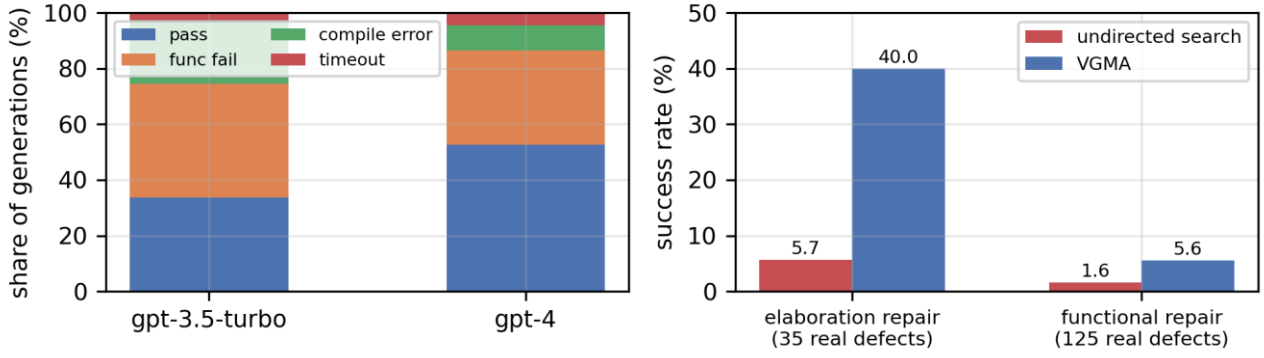

Figure 5. Left: composition of the 220 real GPT-3.5-turbo and GPT-4 generations by verification outcome. Right: success rate on real defects for elaboration repair and for end-to-end functional repair.

5.5. The Equivalence Gate

Bounded sequential equivalence checking was applied to every accepted patch in the undirected and full configurations of Corpus B. Of the 66 patches accepted by full VGMA, 54 (81.8%) were proved equivalent to the reference and 12 were inconclusive within the solver budget; none was refuted. The undirected configuration behaves identically in this respect (27 of 33 proved). Requiring agreement of two independent oracles, one of them a randomised differential testbench, therefore appears to be a strong filter at this design scale, and we did not observe the plausible-but-incorrect patches that dominate software repair [14]. We are careful not to over-claim: the proofs are bounded, the designs are small, and an inconclusive rate near one fifth means the gate is a conservative accept, not a certificate.

5.6. Implementation Quality And Optimisation

Functional correctness is not implementation quality. We synthesised the 75 functionally passing generations and the 19 human references with an identical flow (Table 8, Fig. 6). The median cell-count ratio between an LLM design and its human reference is 1.00, so most passing designs are structurally comparable, but the distribution has a heavy right tail: 25.3% are larger than the reference, 17.3% are at least 20% larger, the 90th percentile is 1.27x and the worst case is 18.8x. Logic depth is much better behaved (mean ratio 1.006, worst 2.0x), which is consistent with area being wasted on redundant state and duplicated arithmetic rather than on longer paths.

The optimisation agent then explored five synthesis scripts on 24 designs. The default flow was already the best-area choice for 16 of them, and the mean improvement of the best script over the default is 2.04% in cells (best case 20.0%) and 1.77% in depth (best case 23.3%). Moreover, for the 8 designs where a non-default script won, the gate could not confirm netlist equivalence within its budget, so the accepted improvement is zero. The conclusion is deliberately negative and useful: at this design scale, tuning the synthesis script is not where the value is. The RTL that the generator emits already costs 27% more area on average than a human implementation, which is an order of magnitude more than the synthesis portfolio can recover, so a PPA-aware generation or rewriting agent is the right place to spend effort, and it must be gated by exactly the kind of equivalence check used here.

6. Discussion And Threats To Validity

6.1. Operator-space Overlap

The injection operators of Corpus B and the repair operators overlap by construction. This does not make the controlled experiment circular, because every configuration searches the same space and the measured quantity is how many queries are needed to find a repair among 42 or more candidates, not whether a repair exists. It does mean the absolute repair rates of Corpus B are optimistic, which is exactly what the comparison with Corpus A demonstrates, and it is why we treat Corpus A as the primary external-validity evidence.

6.2. Bounded Proofs And Tool Limits

Equivalence is proved over a bounded number of cycles from a zero initial state, so "proved" means no counterexample of that length exists, not full sequential equivalence. Seven of 29 RTLLM designs and four of 156 VerilogEval problems could not be verified with our open-source flow and are excluded; a commercial simulator would change these counts. A small number of designs whose single verification query exceeds 0.3 s were excluded from the repair ablation to fit the single-core time budget, which biases absolute rates slightly upward for all configurations equally.

6.3. No LLM in the Loop

The repair backend is deterministic. Consequently our results bound what verification guidance contributes when the patch generator is weak, and they do not measure how much an LLM backend would add; the collapse from 73.3% to 5.6% between corpora is best read as a lower bound on the headroom available to a generative repair agent operating on the same signal. The framework was built so that substituting such a backend changes one adapter, and closing that loop is the obvious next step.

6.4. Scale

Both corpora are module-level. Repository-scale RTL brings hierarchy, parameterisation and cross-module protocol errors, where interface conformance and cone slicing must operate across module boundaries and where bounded equivalence checking becomes considerably more expensive. Benchmarks in that direction have begun to appear [21], and we expect the signature abstraction to transfer while the localisation and gate implementations will need to be re-engineered.

7. Conclusion

We presented VGMA, a verification-guided multi-agent framework for reliable RTL generation and optimisation, and evaluated it entirely with open-source tools on both injected and real defects. Three results stand out. Compressing a failing simulation into a counterexample signature and combining it with a backward cone slice raises top-1 fault localisation from 39.8% to 47.9% and doubles the repair rate under a fixed verification budget, while a differential testbench synthesised automatically from the port interface recovers 92.7% of the defect-detection power of hand-written testbenches. Guidance that does not localise can be worse than no guidance, which argues for designing the verification signal deliberately rather than forwarding raw tool output. And the same framework that repairs 73.3% of single-site injected defects repairs only 5.6% of real LLM defects while fixing 40.0% of real elaboration failures, so the community should expect mutation corpora to overstate end-to-end repair capability substantially, and should invest in patch generators expressive enough to restructure a design rather than in richer search over line-local edits. Finally, a quarter of functionally passing LLM designs are larger than the human reference and the synthesis portfolio recovers only 2.0% of that, which places the next optimisation opportunity in verified RTL rewriting.

Table 8. Post-synthesis quality of functionally passing LLM designs relative to the human reference, and the gain available from a synthesis-script portfolio.

Measurement	Value
Human references / passing LLM designs	19 / 75
Cell ratio, median	1.00x
Cell ratio, mean	1.27x
Cell ratio, 90th percentile	1.27x
Cell ratio, worst case	18.8x
Designs larger than reference	25.3%
Designs at least 20% larger	17.3%
Logic-depth ratio, mean / worst	1.006x / 2.0x
Best script vs default flow, cells	2.04% (max 20.0%)
Best script vs default flow, depth	1.77% (max 23.3%)
Default flow already best (area)	16 of 24 designs

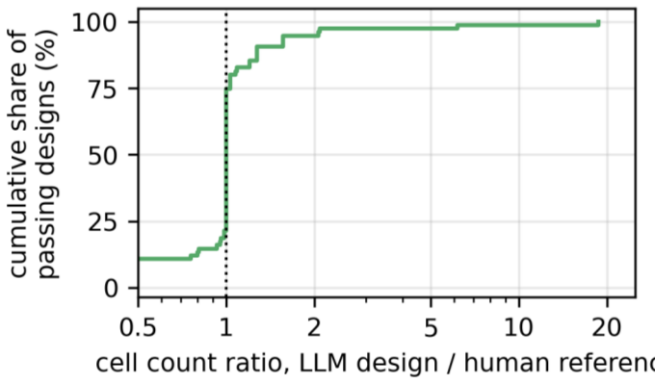

Figure 6. Empirical CDF of the post-synthesis cell-count ratio between a functionally passing LLM design and its human reference (log scale).

References

[1] Liu, M., Pinckney, N., Khailany, B., & Ren, H. (2023). VerilogEval: Evaluating large language models for Verilog code generation. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (pp. 1-8).

[2] Pinckney, N., Batten, C., Liu, M., Ren, H., & Khailany, B. (2024). Revisiting VerilogEval: A year of improvements in large-language models for hardware code generation. arXiv preprint arXiv:2408.11053.

[3] Lu, Y., Liu, S., Zhang, Q., & Xie, Z. (2024). RTLLM: An open-source benchmark for design RTL generation with large language model. In Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC) (pp. 722-727).

[4] Thakur, S., Ahmad, B., Fan, Z., Pearce, H., Tan, B., Karri, R., Dolan-Gavitt, B., & Garg, S. (2023). Benchmarking large language models for automated Verilog RTL code generation. In Proceedings of the Design, Automation and Test in Europe (DATE) (pp. 1-6).

[5] Thakur, S., Ahmad, B., Pearce, H., Tan, B., Dolan-Gavitt, B., Karri, R., & Garg, S. (2024). VeriGen: A large language model for Verilog code generation. ACM Transactions on Design Automation of Electronic Systems, 29(3), 1-31.

[6] Liu, S., Fang, W., Lu, Y., Wang, J., Zhang, Q., Zhang, H., & Xie, Z. (2025). RTLCoder: Fully open-source and efficient LLM-assisted RTL code generation technique. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 44(4), 1448-1461.

[7] Zhao, Y., Huang, D., Li, C., Jin, P., Nan, Z., Ma, T., Qi, L., Pan, Y., Zhang, Z., Zhang, R., Zhang, X., Du, Z., Guo, Q., Hu, X., & Chen, Y. (2024). CodeV: Empowering LLMs with HDL generation through multi-level summarization. arXiv preprint arXiv:2407.10424.

[8] Thakur, S., Blocklove, J., Pearce, H., Tan, B., Garg, S., & Karri, R. (2023). AutoChip: Automating HDL generation using LLM feedback. arXiv preprint arXiv:2311.04887.

[9] Ho, C.-T., Ren, H., & Khailany, B. (2025). VerilogCoder: Autonomous Verilog coding agents with graph-based planning and abstract syntax tree based waveform tracing tool. In Proceedings of the AAAI Conference on Artificial Intelligence.

[10] Zhao, Y., Zhang, H., Huang, H., Yu, Z., & Zhao, J. (2025). MAGE: A multi-agent engine for automated RTL code generation. In Proceedings of the ACM/IEEE Design Automation Conference (DAC).

[11] Wei, Y., Huang, Z., Li, H., Xing, W. W., Lin, T.-J., & He, L. (2025). VFlow: Discovering optimal agentic workflows for Verilog generation. arXiv preprint arXiv:2504.03723.

[12] Tsai, Y.-D., Liu, M., & Ren, H. (2024). RTLFixer: Automatically fixing RTL syntax errors with large language models. In Proceedings of the ACM/IEEE Design Automation Conference (DAC).

[13] Smith, E. K., Barr, E. T., Le Goues, C., & Brun, Y. (2015). Is the cure worse than the disease? Overfitting in automated program repair. In Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE) (pp. 532-543).

[14] Qi, Z., Long, F., Achour, S., & Rinard, M. (2015). An analysis of patch plausibility and correctness for generate-and-validate patch generation systems. In Proceedings of the International Symposium on Software Testing and Analysis (ISSTA) (pp. 24-36).

[15] Zhao, W., Chen, T., Yang, J. S., & Qiu, L. (2026). AutoML-Pipeline: A RAG-enhanced code generation framework with pre-validation for cloud-native machine learning workflows. IEEE Access, 14, 41932-41945.

[16] Williams, S., & Baxter, M. (2002). Icarus Verilog: Open-source Verilog more than a year later. Linux Journal, 99.

[17] Wolf, C., & Glaser, J. (2013). Yosys: A free Verilog synthesis suite. In Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip) (pp. 47-52).

[18] Cui, F., Yin, C., Zhou, K., Xiao, Y., Sun, G., Xu, Q., Guo, Q., Liang, Y., Zhang, X., Song, D., & Others. (2024). OriGen: Enhancing RTL code generation with code-to-code augmentation and self-reflection. In Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD) (pp. 1-9).

[19] Yang, Y., Teng, F., Liu, P., Qi, M., Lv, C., Li, J., Zhang, X., & He, Z. (2025). HaVen: Hallucination-mitigated LLM for Verilog code generation aligned with HDL engineers. In Proceedings of the Design, Automation and Test in Europe (DATE) (pp. 1-7).

[20] Wang, Y., Sun, G., Ye, W., Qu, G., & Li, A. (2025). VeriReason: Reinforcement learning with testbench feedback for reasoning-enhanced Verilog generation. arXiv preprint arXiv:2505.11849.

[21] Pinckney, N., Deng, C., Ho, C.-T., Tsai, Y.-D., Liu, M., Zhou, W., Khailany, B., & Ren, H. (2025). 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.

[22] Ahmad, H., Huang, Y., & Weimer, W. (2022). CirFix: Automatically repairing defects in hardware design code. In Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS) (pp. 990-1003).

[23] Le Goues, C., Nguyen, T., Forrest, S., & Weimer, W. (2012). GenProg: A generic method for automatic software repair. IEEE Transactions on Software Engineering, 38(1), 54-72.

[24] Xu, B., Ma, Y., Wang, C., Xia, Y., & Others. (2024). Location is key: Leveraging large language model for functional bug localization in Verilog. arXiv preprint arXiv:2409.15186.

[25] Vasudevan, S., Sheridan, D., Patel, S., Tcheng, D., Tuohy, B., & Johnson, D. (2010). GoldMine: Automatic assertion generation using data mining and static analysis. In Proceedings of the Design, Automation and Test in Europe (DATE) (pp. 626-629).

[26] Jones, J. A., & Harrold, M. J. (2005). Empirical evaluation of the Tarantula automatic fault-localization technique. In Proceedings of the 20th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 273-282).

[27] Abreu, R., Zoeteweij, P., & van Gemund, A. J. C. (2006). An evaluation of similarity coefficients for software fault localization. In Proceedings of the 12th Pacific Rim International Symposium on Dependable Computing (PRDC) (pp. 39-46).

[28] Abreu, R., Zoeteweij, P., & van Gemund, A. J. C. (2007). On the accuracy of spectrum-based fault localization. In Proceedings of Testing: Academic and Industrial Conference Practice and Research Techniques (TAICPART-MUTATION) (pp. 89-98).

[29] Wu, J., Zhang, Z., Yang, D., Meng, X., He, J., Mao, X., & Lei, Y. (2022). Fault localization for hardware design code with time-aware program spectrum. In Proceedings of the IEEE International Conference on Computer Design (ICCD) (pp. 537-544).

[30] Mo, T., Zhang, C., Zou, J., Guo, Z., & Rhee, M. (2026). Self-evolving AI agents with dual memory for automated software testing and bug localization. IEEE Access, 14, 111086-111102.

[31] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing reasoning and acting in language models. In Proceedings of the International Conference on Learning Representations (ICLR).

[32] Wu, Q., Bansal, G., Zhang, J., Wu, Y., Zhang, S., Zhu, E., Li, B., Jiang, L., Zhang, X., & Wang, C. (2023). AutoGen: Enabling next-gen LLM applications via multi-agent conversation framework. arXiv preprint arXiv:2308.08155.

[33] Yang, J., Jimenez, C. E., Wettig, A., Lieret, K., Yao, S., Narasimhan, K., & Press, O. (2024). SWE-agent: Agent-computer interfaces enable automated software engineering. In Advances in Neural Information Processing Systems (Vol. 37, pp. 50528-50652).

[34] Rhee, M., Zou, J., Mo, T., Teng, D., & Yang, J.-S. (2026). Enhancing web search agents with self-play contrastive fine-tuning. IEEE Access. https://doi.org/10.1109/ACCESS.2026.3717594

[35] Zhang, H., Ge, Y., Zhao, X., & Wang, J. (2025). Hierarchical deep reinforcement learning for multi-objective integrated circuit physical layout optimization with congestion-aware reward shaping. IEEE Access, 13, 162533-162551.

[36] Brayton, R., & Mishchenko, A. (2010). ABC: An academic industrial-strength verification tool. In Proceedings of the International Conference on Computer Aided Verification (CAV) (pp. 24-40).