TopoScope: A Graph-Scoped LLM Agent for Printed Circuit Board Schematic Design

Jiale Guo, Shiji Tong, Weihao Li, Yu Wu, Boxian Lin, Mengji Shi*
University of Electronic Science and Technology of China
jialeeguo@gmail.com

Abstract

Printed circuit boards (PCBs) are the hardware foundation of modern electronic systems. However, automating PCB schematic design remains difficult. A valid schematic must respect real-IC pin-level constraints and cross-component topological relations, neither of which is captured by generic simulation, local rule checks, or direct LLM generation. To address this challenge, we present TopoScope, a training-free LLM agent that uses the evolving schematic graph to control both generation and verification. Within this agent, Topology-Guided Context Scoping determines the entities visible for each component-level wiring decision, while Pattern-Based Rule Verification checks generated subgraphs against datasheet-derived constraints. These complementary mechanisms operate over one shared graph, whose pattern-matching substrate also supports post-generation functional evaluation. We further introduce a 20-design benchmark spanning three complexity tiers and six application domains, with an assertion-based evaluation protocol calibrated against expert judgments. Across seven recent LLMs, TopoScope improves functional correctness, reduces invalid graph references, and enables more reliable generation of medium and hard multi-IC schematics compared with knowledge-provided baselines. Code is available at https://github.com/idokira/TopoScope.

1 Introduction

Printed circuit boards (PCBs) are the hardware foundation of modern electronic systems. Every device that contains silicon, from consumer electronics to automotive and industrial systems [1], goes through a PCB design flow. In this flow, heterogeneous integrated circuits such as digital controllers, analog front-ends, power converters, and communication transceivers are combined into a working board. At the center of this process is the schematic: an interconnection graph that specifies how the pins of real components are connected to realize the intended electrical behavior, and that serves as the interface between chip-level design and board-level layout [2].

Despite its central role, schematic design remains highly manual, for four interrelated reasons. First, its building blocks are real ICs rather than idealized devices. A valid design must follow manufacturer-defined pin behavior and application constraints, yet no universal physical model predicts whether an arbitrary composition of heterogeneous chips will function, and SPICE-class simulation does not capture many of the system-level constraints that govern real components. Second, correctness is tied to pin-level physical reality. Unlike stylistic issues in text generation, an incorrect pin identity or rail assignment can translate directly into hardware-level failure. Third, correctness depends on cross-cutting topological constraints such as voltage-domain consistency, ground isolation, feedback completeness, and bypass-capacitor placement. These constraints span many components and nets rather than any single wire. Fourth, the field lacks the infrastructure that enabled automation in neighboring domains such as digital circuit design. Unlike digital RTL, PCB schematic design has no widely adopted description language comparable to Verilog, no general-purpose simulator

*Corresponding author.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

that covers heterogeneous IC compositions end-to-end, and no open corpus at board-level scale [3]. Existing public circuit datasets mainly focus on analog or mixed-signal settings [4–6], rather than heterogeneous PCB designs built from real ICs. Together, these factors make schematic design difficult for both simulation-driven and purely data-driven automation. The challenge does not lie in any single decision; it lies in how many local decisions combine into a valid topology.

To address this structural challenge, we present TopoScope, a training-free LLM agent that uses the evolving schematic graph to control both generation and verification. It integrates Topology-Guided Context Scoping (TCS), which determines the entities visible for each component-level decision, and Pattern-Based Rule Verification, which checks generated subgraphs against datasheet-derived constraints. TCS and verification are complementary mechanisms within the overall agent, operating over the same evolving graph. The shared pattern-matching substrate supports both in-loop feedback and post-generation functional evaluation. We further introduce a 20-design benchmark across three complexity tiers and six application domains, with an assertion-based evaluation protocol calibrated against expert judgments. Our contributions are summarized as follows:

- **Topology-guided context scoping.** We introduce TCS, a runtime mechanism that uses the hierarchical structure and current state of the schematic graph to compute a scoped view for each component-level wiring decision. The model is exposed only to the pins, active nets, global nets, and module interfaces relevant to the current action, while out-of-scope entities cannot be referenced by construction.
- **A unified graph-pattern substrate.** We formulate datasheet-derived electrical rules and design-level functional assertions as graph patterns over the schematic. The same pattern-matching substrate supports in-loop feedback during generation and fine-grained functional evaluation after generation, grounded by a component knowledge base containing per-IC pin definitions, reference circuits, and electrical rules.
- **A PCB benchmark and evaluation methodology.** We construct a benchmark of 20 designs spanning three complexity tiers and six application domains, and develop an assertion-based protocol for measuring schematic correctness at fine granularity. We calibrate the protocol against expert judgments and use it to evaluate seven recent LLMs and ablate the proposed generation mechanisms.

2 Related Work

2.1 Circuit Design Automation

Circuit design automation spans multiple abstraction levels, and the degree of automation varies substantially across them. At the register-transfer level, digital synthesis benefits from standardized hardware description languages and mature open-source toolchains, where simulation through testbenches and formal methods provides an established verification foundation.

Analog circuit design has been a long-standing target of learning-based automation, with effort distributed across topology generation, device sizing, and dataset construction. Prior work has explored two-level GNN models for joint topology generation and device sizing [4], language-modeling and generative formulations of analog topology synthesis [6, 7], Bayesian-optimization-based sizing methods [8], and AMS netlist datasets for LLM-based circuit design [5].

For PCB design, automation has largely targeted placement and routing rather than schematic synthesis. Prior work has applied learning-based methods to fan-out routability [9] and to GNN-based component addition for PCB schematics [10]. Commercial schematic tools such as KiCad, Altium, and Cadence OrCAD provide pin-level electrical rule checking via compatibility matrices over pin types, which catches local inconsistencies. Schematic-level generation has seen limited traditional automation, partly because heterogeneous IC compositions do not admit a uniform device model.

2.2 LLM-based Approaches

The rise of large language models has opened a new direction for circuit automation. We organize recent LLM-based work along the abstraction ladder, from early device-level studies to digital, analog, and PCB-level approaches. As an early study, From Words to Wires [11] showed that contemporary LLMs can produce component lists, pinouts, netlist-style schematics, and microcontroller code

directly from natural language, introducing the PINS100 and MICRO25 benchmarks for component-level knowledge and Arduino-ecosystem circuits. At the digital RTL level, LLM-based systems benefit from mature HDL toolchains, where generated code can be compiled, simulated, and repaired through compiler or testbench feedback. Prior work has explored conversational co-design up to tapeout [12], targeted Verilog generation and unified understanding-generation models [13, 14], domain-adapted chip-engineering assistants [15], autonomous RTL-to-GDSII scripting [16], and LLM-driven AI accelerator design [17]. Closed-loop systems such as AutoChip [18] and LLM4Netlist [19] further pair generation with tool feedback for iterative repair.

LLM-based analog circuit design has produced a substantial body of agent-style work, distributed across SPICE-in-the-loop generation, knowledge-augmented retrieval, and multi-stage agent workflows. Prior systems have explored training-free agents that pair PySpice generation with SPICE feedback [20], knowledge-graph-augmented retrieval coupled with Bayesian-optimization sizing [5], multi-agent workflows spanning specification understanding, testbench generation, and RF/analog netlist design [21], multimodal agents for analog schematic generation [22], and large-scale SPICE netlist datasets curated through LLM-driven workflows [23]. A separate line treats topology discovery as a generative problem, training a GPT-style engine over sequence-based pin-level graph representations of analog topologies [6]. At the PCB level, LLM-based work has begun to enter layout and schematic tasks. Prior efforts include multi-LLM agent collaboration for analog layout [24] and reinforcement-learning-driven LLMs for circuit topology exploration [25]. For schematic generation, CircuitLM [26] develops a multi-agent pipeline over CircuitJSON synthesis with hybrid ERC-and-LLM-judge evaluation, PCBSchemaGen [27] grounds SKiDL code generation in a datasheet-derived knowledge graph with post-generation subgraph-isomorphism verification, and Schemato [28] targets LLM-based netlist-to-schematic conversion. HWE-Bench [3] introduces a board-level schematic design benchmark with rule-based and simulation-oriented evaluation.

Overall, prior LLM-based circuit design often combines grounded component knowledge, structured representations, and feedback from external tools. PCB schematic generation remains harder because correctness depends on chip-specific pin roles and cross-component topology over heterogeneous real ICs. TopoScope uses the schematic graph during generation: datasheet-derived graph patterns shape both the scoped context and the local feedback for each component-level wiring decision.

3 Method

TopoScope is an LLM agent for PCB schematic design, organized around a single data structure, the schematic graph, and coupled with two mechanisms that operate on it: Topology-Guided Context Scoping (TCS) decides what the model sees at each step, and Pattern-Based Rule Verification expresses electrical rules as graph patterns and checks them on the same graph. Figure 1 shows the overall workflow. Given a design task, a global planning step decomposes the board into functional modules and declares the signals that cross module boundaries as virtual ports (Figure 1a). Modules are then processed in traversal order. Within a module, each component is handled by a single LLM call that emits its full connection set in one structured response (Figure 1b), grounded on two inputs: a datasheet-derived standard specification supplying pin roles, reference circuit, and electrical rules, and a compressed summary of already-processed modules. The emitted connections are merged into the graph and verified against the component's rules; on failure, the component is regenerated under the same scoped view for a total budget of six attempts. Once all modules are complete, a merging step resolves virtual-port references into ordinary nets, yielding the final netlist.

Formally, the input and output are

$$\mathcal{D} = (s, C), \qquad G_{\text{out}} = \text{TopoScope}(\mathcal{D}, \text{KB}_C), \tag{1}$$

where s specifies functionality and connectivity, C lists pre-selected IC instances with exact part numbers, and KB_C supplies their pin definitions, reference circuits, and electrical rules. Component selection is upstream; generation assumes accurate records covering the selected parts. The specification must state relevant supply, interface, and design constraints, without requiring fixed wording. The output is a pin-level schematic graph, not PCB placement or routing.

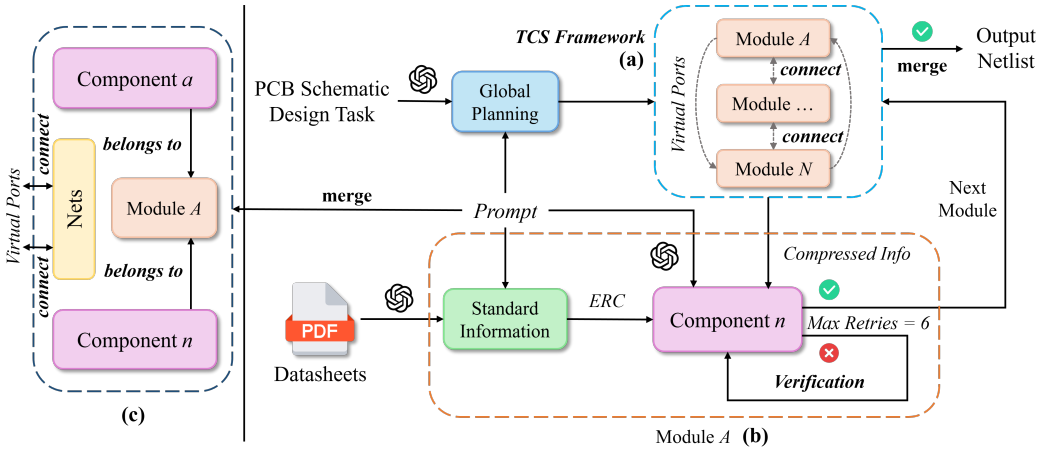


Figure 1: Overview of TopoScope. (a) Global planning decomposes the design into modules and declares cross-module signals as virtual ports, resolved into ordinary nets in a final merging step. (b) Each component is processed by a scoped LLM call grounded on a datasheet-derived standard specification and a compressed summary of already-processed modules; emitted connections are verified before the loop advances, with a six-attempt budget. (c) Components, nets, and virtual-port references form the graph substrate on which scoping and verification operate; detailed in Figure 2.

3.1 Schematic Graph

At its electrical core, a schematic records one fact: which pins share the same net. Pins connect only through nets, making the electrical layer a bipartite graph

$$B = (V_{\text{pin}},\ V_{\text{net}},\ E_{\text{connects}}) \tag{2}$$

with two disjoint vertex sets and edges only between them. A final netlist, stripped of drawing conventions, is exactly an instance of this graph. The bipartite layer is complete but flat: presenting several hundred pins and dozens of nets to the LLM at once dilutes attention and invites cross-wiring. It records what is connected, but not which pins and nets are locally relevant at a given generation step.

To make locality explicit, we augment the bipartite core with an organizational layer. Two node types, module and component, partition pins into nested groups via a second type of edge, *belongs*: a pin belongs to a component, a component belongs to a module, and modules form the top level of the hierarchy (Figure 2a). Organizational nodes carry no electrical meaning and never appear on *connects* edges; they exist only to make locality a first-class property of the graph rather than a heuristic recovered from names. The complete graph is then

$$G = (V, E), \quad V = V_{\text{pin}} \cup V_{\text{net}} \cup V_{\text{comp}} \cup V_{\text{mod}}, \quad E = E_{\text{connects}} \cup E_{\text{belongs}}, \tag{3}$$

with the two edge types playing orthogonal roles. *Connects* edges are restricted to pin-to-net pairs and carry all electrical meaning. *Belongs* edges form a hierarchical forest over pin-to-component, component-to-module, and net-to-module relations. Each pin participates in at most one *connects* edge, which enforces the single-net-per-pin property of a valid netlist.

The electrical output B_{out} is the pin–net subgraph of G_{out}. Structural validity requires legal component and pin references and at most one net per pin, with virtual ports resolved before return. These conditions do not establish functional correctness.

Each pin carries an electrical type (power, input, output, analog, bidirectional, open-drain, etc.) that grounds the subgraph patterns in Section 3.3, and each net carries a net class and a short semantic role assigned at creation, so already-created nets are self-describing when they later appear in another component's scoped view. Global nets such as GND and VCC are shared across the design, and cross-module signals are represented as virtual-port references (Section 3.2) rather than persistent nodes.

To populate this graph, we maintain a component knowledge base: for each part number, a JSON document extracted once from the manufacturer's datasheet by an LLM pipeline. Each document supplies pin definitions (used to instantiate pin nodes and *belongs* edges, so the model never recalls

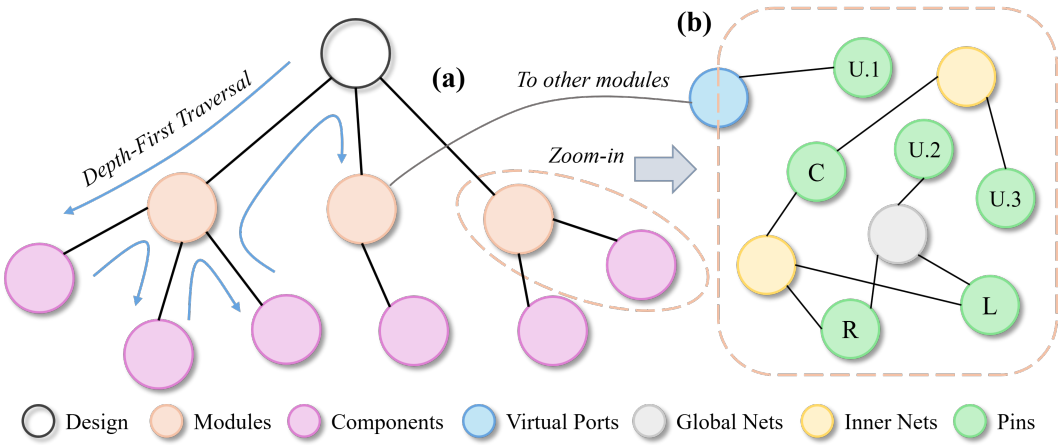


Figure 2: Schematic graph and traversal. (a) The organizational layer forms a hierarchy from design to module to component to pin; components are visited depth-first (blue arrows; detailed in Section 3.2). (b) A module in detail: the focused component (U.1–U.3) exposes all of its pins, nearby passives (R, L, C) contribute attachment pins, pins attach to inner nets or global nets, and a virtual port denotes a cross-module net reference to be resolved at merging time.

a pinout), a reference circuit giving the manufacturer's typical application topology as a template the model adapts, and electrical rules expressed as graph patterns that drive in-loop verification (Section 3.3). Adding support for a new chip amounts to adding one JSON document.

3.2 Topology-Guided Context Scoping

Topology-Guided Context Scoping (TCS) computes, for each component, a scoped view of the schematic graph that contains only the pins, nets, and cross-module references relevant to wiring that component. At the moment a component c is processed inside module m, TCS exposes to the model the scoped view $\mathcal{V}(c, m)$,

$$\mathcal{V}(c,m) = \underbrace{\mathrm{pins}(c) \cup \mathrm{ref}(c) \cup \mathrm{rules}(c)}_{\text{focus layer}} \cup \underbrace{N_m^{\mathrm{active}} \cup \mathrm{VP}(m) \cup N_{\mathrm{global}}}_{\text{environment layer}}, \tag{4}$$

organized into two layers.

Focus layer. This layer is fixed once c is known: it contains c's pins with their electrical types, the datasheet-derived reference circuit $\mathrm{ref}(c)$, and the electrical rules $\mathrm{rules}(c)$ that c must satisfy.

Environment layer. This layer carries the attachment points that c is allowed to connect to: N_m^{active} is the set of nets already created inside m by earlier components, $\mathrm{VP}(m)$ is the set of virtual-port references declared for m at global planning time (each representing a signal that will connect m to another module), and N_{global} contains the power and ground nets shared across the design. Figure 2(b) illustrates this view from the perspective of a component in focus.

In prompt composition the focus layer dominates, the environment layer is added next, and any remaining space holds a compressed summary of already-processed modules. Given this view, the LLM emits c's connection set in one structured response: for each pin of c, an attachment to an existing visible net or the creation of a new net, together with any local passives implied by $\mathrm{ref}(c)$.

At attempt k, this generation interface is

$$p_k = \textsc{BuildPrompt}(\mathcal{D}, \mathcal{V}(c,m), \mathcal{H}, \mathcal{E}_{k-1}), \qquad J_k = \mathrm{LLM}(p_k), \tag{5}$$

where $\mathcal{H}$ is the compressed summary when space permits, $\mathcal{E}_{k-1}$ is the previous attempt's rule-violation feedback, and $\mathcal{E}_0 = \emptyset$. The structured candidate J_k may reference visible existing endpoints or explicitly create local nets and required passives; it may not invent IC pin identifiers or reference hidden graph entities. Prompt templates are given in Appendix A.6.

The view depends on the state of the graph when c is processed, so the order in which components are visited determines what each view contains. TopoScope traverses the organizational layer in depth-first order (Figure 2a): modules are processed one at a time, and within each module components

are processed in sequence. Inside module m, N_m^{active} grows monotonically. It is empty when m is first entered, and after a component's output is selected, its new nets, local passives, and attachments are committed to the graph and become part of N_m^{active} for all later components in m. Verification determines this selection, either by accepting a rule-satisfying candidate or by choosing the lowest-error-rate candidate when the attempt budget is exhausted (Section 3.3). When the traversal leaves m for the next module, a fresh active set begins, and inner nets do not leak across module boundaries.

Cross-module signals are not exposed through N_m^{active}. They are carried instead by the virtual ports declared at global planning time. A pin in m may attach to an element of $\mathrm{VP}(m)$ in the same way it attaches to an ordinary net, but at the level of the graph these references are transient: they are not elements of V_{net}, carry no net class, and do not appear in any module's active set. From the model's perspective, a virtual port is a named endpoint that other modules will also attach to on their own side. After all modules have been processed, a merging step resolves virtual-port references into ordinary nets: references that share the same declared identity across modules are collapsed into a single net, which inherits a net class from the pins that now attach to it. The result is a single connected netlist in which no virtual-port reference remains.

Together, the scoped view and the structured output format bound what the model can express: existing entities outside $\mathcal{V}(c, m)$ cannot be referenced by construction, which yields a structural form of hallucination containment. This restriction does not prohibit explicitly creating new local nets within the current candidate.

3.3 Pattern-Based Rule Verification

Most chip-level electrical requirements span multiple pins, multiple nets, and the passive components linking them. A bypass capacitor requirement, for example, asserts the existence of a small structured subgraph: a power pin attaches to a net, that net also attaches to a capacitor, and the capacitor's other side attaches to ground. The requirement is a property of a small fragment of the graph, and a pairwise pin-type compatibility check cannot express this kind of multi-node constraint. TopoScope therefore expresses each electrical rule as a graph pattern over the schematic graph.

Definition 1 (Pattern Graph). A pattern is a labelled graph $P = (V_P, E_P, \lambda)$, where λ assigns to each node and edge a label specifying the required type and the attribute conditions it must satisfy.

Verification of a rule then reduces to deciding whether the pattern is realized in the current schematic graph at the rule's anchor pin.

Definition 2 (Pattern Satisfaction). Given the schematic graph G, a pattern P, and an anchor pin $a \in V_{\text{pin}}$, P is satisfied at a in G if there exists an injective mapping $\phi : V_P \to V(G)$ such that $\phi(v_P^{\text{anchor}}) = a$; for every $(u, v) \in E_P$, $(\phi(u), \phi(v)) \in E(G)$; and for every $x \in V_P \cup E_P$, $\lambda(x) \models \mathrm{label}_G(\phi(x))$.

This is a label-constrained subgraph isomorphism on the local neighborhood of a. The anchor constraint pins the search to a fixed point in G, so the matching cost depends on the size of a's neighborhood rather than on the size of the full graph, and each pattern in our library is small (under six nodes).

The patterns that arise in PCB design fall into three families, two chip-level and one system-level: **adjacency patterns** over a one- or two-hop neighborhood of the anchor (such as a bypass capacitor on a power pin, orange box in Figure 3), **path patterns** over a bounded-length path between two named anchors (such as a feedback divider between an output and a feedback pin, blue box in Figure 3), and **system-level patterns** over attributes that propagate along nets across the full graph (such as voltage-domain consistency and analog-digital ground separation). The two chip-level families are written once per component in the knowledge base and checked during generation; system-level patterns are well-defined only on the fully merged graph and are checked externally.

The chip-level families participate directly in the generation pipeline. After the LLM emits a connection set J for the focused component c, every adjacency and path pattern in $\mathrm{rules}(c)$ is checked on the graph tentatively extended with J; on failure, the component is regenerated under the same scoped view with the previous violations supplied as feedback, for up to six attempts. Algorithm 1 makes this loop precise.

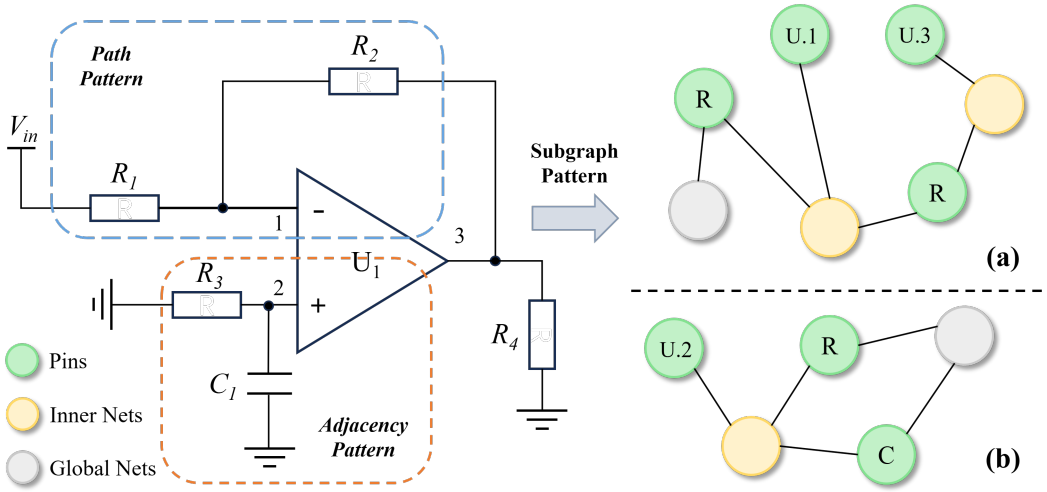


Figure 3: Electrical rules as subgraph patterns. A schematic fragment (left) is mapped to its underlying graph (right); each rule corresponds to a small subgraph pattern anchored at a designated pin. The blue box marks a path pattern (panel a) and the orange box marks an adjacency pattern (panel b). Verification reduces to subgraph matching (Definition 2).

Algorithm 1 Scoped Schematic Generation with Pattern Verification

Require: design specification $\mathcal{D} = (s, C)$, component records KB_C
Ensure: schematic graph G

$(M, VP) \leftarrow \textsc{GlobalPlan}(\mathcal{D})$ ▷ M: modules, VP: virtual ports
$G \leftarrow \textsc{InitGraph}(M, \mathrm{KB}_C)$ ▷ pin nodes and *belongs* edges
for each $m \in M$ in DFS order **do**
  $N_m^{\text{active}} \leftarrow \emptyset$
  for each $c \in m$ in DFS order **do**
    $\mathcal{V} \leftarrow \textsc{FocusLayer}(c) \cup \textsc{EnvLayer}(N_m^{\text{active}}, VP(m), N_{\text{global}})$
    $\mathcal{H} \leftarrow \textsc{CompressedSummary}(G)$; $\mathcal{E}_0 \leftarrow \emptyset$; $\mathcal{J} \leftarrow \emptyset$
    for $k = 1$ to 6 **do**
      $J_k \leftarrow \text{LLM}(\textsc{BuildPrompt}(\mathcal{D}, \mathcal{V}, \mathcal{H}, \mathcal{E}_{k-1}))$; $\mathcal{J} \leftarrow \mathcal{J} \cup \{J_k\}$
      $\mathcal{E}_k \leftarrow \{\, r \in \text{rules}(c) : \neg\, \textsc{Match}(P_r, G \cup J_k, \text{anchor}_r) \,\}$
      if $\mathcal{E}_k = \emptyset$ **then**
        break ▷ first rule-satisfying candidate
      end if
    end for
    Select $J^\star \in \arg\min_{J \in \mathcal{J}} \rho_c(J)$ ▷ lowest error rate; zero if a candidate passes
    $(G, N_m^{\text{active}}) \leftarrow \textsc{Commit}(G, N_m^{\text{active}}, J^\star)$
  end for
end for
$G \leftarrow \textsc{ResolveVPs}(G, VP)$
return G

For a fixed pre-commit graph G, the verifier returns

$$\mathcal{E}_k = \{r \in \text{rules}(c) : \neg\, \textsc{Match}(P_r, G \cup J_k, \text{anchor}_r)\}. \tag{6}$$

A candidate with $\mathcal{E}_k = \emptyset$ is committed immediately; otherwise these violations are returned as repair feedback. If all six attempts fail, we select $J_{k^\star}$, where $k^\star \in \arg\min_{1 \le k \le 6} \rho_c(J_k)$ and ρ_c denotes the local verifier's error rate, and commit it as the component's output. This fallback retains the best candidate under the checking budget, not a guarantee that all rules pass; final task-level evaluation remains independent of this local decision.

Table 1: Summary of benchmark tasks.

ID	Type	Task Summary	ID	Type	Task Summary
1	COM	USB-to-UART bridge with on-board LDO supply	2	PM	Fixed-output LDO with input and output bulk capacitors
3	LED	Constant-current LED driver with current-set resistor	4	AUD	Single-supply audio power amplifier with gain network
5	SEN	MCU with shared I2C bus to sensor and OLED	6	SEN	MCU reading I2C IMU and logging to SPI Flash
7	COM	MCU with USB-to-UART bridge and I2C EEPROM	8	COM	MCU to CAN controller over SPI with transceiver
9	COM	MCU with RS485 transceiver and SPI Flash logging	10	PM	Triple-rail supply with comparator-based power-good
11	DRV	MCU-driven H-bridge with shunt-based current sensing	12	AUD	Op-amp preamp with I2C DAC bias control
13	LED	SPI LED matrix driver cascaded with shift register	14	SEN	I2C ADC with external precision voltage reference
15	SEN	Flight controller with separate digital and analog 3.3V rails, I2C IMU, shared SPI bus for barometer and Flash	16	COM	Battery-powered IoT gateway with Li-ion charger, protection, and WiFi MCU
17	DRV	Dual stepper drivers over UART with SPI parameter Flash and dual-rail regulation	18	AUD	Stereo audio chain with boost-supplied preamp and class-D amplifier
19	PM	Branched multi-rail supply with buck, boost, cascaded LDOs, and power-good supervisor	20	SEN	Galvanically isolated precision ADC with digital isolator across SPI

4 Experiments

4.1 Experimental Setup

Models. We evaluate on seven recent large language models spanning different providers and model families: GPT-5.4 [29], Gemini-3.1-Pro [30], Qwen3.5-Plus [31], GLM-5 [32], Kimi-K2.5 [33], DeepSeek-V3.2 [34], and GPT-OSS-120B [35], included as a mid-scale model representative of potential locally-hosted deployments. We use provider-default decoding settings and do not tune sampling hyperparameters. All models and conditions share the same prompt up to condition-specific instructions, detailed in Appendix A.6.

Benchmark. To evaluate PCB schematic generation end to end, we construct a benchmark whose outputs are normalized to a common schematic-graph representation, so that all methods are judged on the same pin-level and topology-level objects. The benchmark contains 20 designs organized along two axes: circuit complexity and application domain. Complexity spans three levels, easy, medium, and hard, corresponding to a single integrated circuit with its required passives up to multi-module designs coordinating five to six integrated circuits. For the tier-level comparisons in Tables 3 and 4, the approximate design sizes are fewer than 60 pins (Easy), 60–120 pins (Medium), and 120–200 pins (Hard); difficulty also reflects IC count and inter-component topology. The domain axis covers six categories: power management, sensing, communication, motor and actuator drive, audio, and LED and display. Table 1 lists all 20 designs with identifier, domain, and task summary; per-design statistics and the full specifications are in Appendix A.1.

Evaluation Metrics. Each design is accompanied by a set of functional assertions that capture system-level design intent. Each assertion is a subgraph pattern paired with a pass condition, evaluated on the generated circuit's schematic graph using the same matching substrate as the in-loop rules but authored separately to encode functional intent rather than chip-level electrical constraints (specification in Appendix A.4). We report two metrics. Pass@k is computed per design from $n = 7$ runs with the unbiased estimator $\text{Pass@}k = 1 - \binom{n-c}{k} / \binom{n}{k}$ [36], where c is the number of passing runs. Assertion Pass Rate (APR) is the fraction of assertions passed in a given run. A run counts as a pass when it is structurally valid, no pin is assigned to more than one net, and its APR meets an expert-calibrated threshold (tier-specific thresholds: 0.800 on easy, 0.890 on medium, and 0.900 on hard). We report Pass@1, Pass@3, and mean APR.

Task-level assertions $\mathcal{A}_d$ are evaluation targets, not generation-prompt inputs: $\text{APR}(G, d) = |\mathcal{A}_d|^{-1} \sum_{a \in \mathcal{A}_d} F_a(G)$, where $F_a(G) \in \{0, 1\}$. Final success requires structural validity and $\text{APR}(G, d) \geq \theta_{\text{tier}(d)}$, not merely local in-loop acceptance.

Human Expert Evaluation. We validate the assertion-based protocol against domain-expert labels using Cohen's κ [37], and calibrate the APR threshold by sweeping $\theta \in [0.8, 1.0]$ to maximize κ. The expert study, calibration procedure, and threshold-calibration curves are in Appendix A.8.

4.2 Main Results

Baseline. We use a refined-knowledge baseline setting: each model receives the design specification, the pin definitions and datasheet reference entries for all involved components, and the naming

Table 2: Main results. Models are sorted left-to-right by overall Pass@1 (worst → best).

Model	DeepSeek-V3.2		GPT-OSS-120B		Kimi-K2.5		Qwen3.5-Plus		GLM-5		GPT-5.4		Gemini-3.1-Pro	
Task ID	Pass@1	Pass@3	Pass@1	Pass@3	Pass@1	Pass@3	Pass@1	Pass@3	Pass@1	Pass@3	Pass@1	Pass@3	Pass@1	Pass@3
1	**100.0**	**100.0**	**100.0**	**100.0**	57.1	97.1	**100.0**	**100.0**	85.7	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
2	85.7	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
3	71.4	**100.0**	**100.0**	**100.0**	71.4	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	85.7	**100.0**	**100.0**	**100.0**
4	42.9	88.6	**100.0**	**100.0**	14.3	42.9	85.7	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
5	42.9	88.6	71.4	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	85.7	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
6	0.0	0.0	57.1	97.1	85.7	**100.0**	**100.0**	**100.0**	14.3	42.9	85.7	**100.0**	**100.0**	**100.0**
7	14.3	42.9	42.9	88.6	28.6	71.4	28.6	71.4	85.7	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
8	57.1	97.1	71.4	**100.0**	85.7	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
9	0.0	0.0	20.0	60.0	57.1	97.1	71.4	**100.0**	57.1	97.1	71.4	**100.0**	**100.0**	**100.0**
10	28.6	71.4	28.6	71.4	28.6	71.4	71.4	**100.0**	**100.0**	**100.0**	71.4	**100.0**	71.4	**100.0**
11	0.0	0.0	42.9	88.6	14.3	42.9	57.1	97.1	85.7	**100.0**	85.7	**100.0**	42.9	88.6
12	14.3	42.9	28.6	71.4	57.1	97.1	14.3	42.9	57.1	97.1	14.3	42.9	28.6	71.4
13	0.0	0.0	16.7	50.0	14.3	42.9	85.7	**100.0**	**100.0**	**100.0**	71.4	**100.0**	**100.0**	**100.0**
14	14.3	42.9	28.6	71.4	85.7	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**	71.4	**100.0**	**100.0**	**100.0**
15	28.6	71.4	0.0	0.0	0.0	0.0	57.1	97.1	**100.0**	**100.0**	85.7	**100.0**	**100.0**	**100.0**
16	0.0	0.0	0.0	0.0	0.0	0.0	14.3	42.9	28.6	71.4	**100.0**	**100.0**	14.3	42.9
17	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	57.1	97.1	28.6	71.4	85.7	**100.0**
18	71.4	**100.0**	60.0	**100.0**	28.6	71.4	71.4	**100.0**	**100.0**	**100.0**	57.1	97.1	85.7	**100.0**
19	42.9	88.6	**100.0**	**100.0**	71.4	**100.0**	71.4	**100.0**	85.7	**100.0**	85.7	**100.0**	**100.0**	**100.0**
20	0.0	0.0	0.0	0.0	28.6	71.4	28.6	71.4	0.0	0.0	85.7	**100.0**	71.4	**100.0**
# Solved	13	13	16	16	17	17	19	19	19	19	20	20	20	20
Average	30.7	–	48.4	–	46.4	–	67.9	–	77.1	–	80.0	–	85.0	–

conventions, then produces the schematic in a single forward pass. This isolates TopoScope's generation procedure from the contribution of component knowledge itself, since the model has access to the same knowledge entries that the full system queries through TCS, but without topology-guided context scoping or in-loop pattern-based verification. We run the baseline on three representative models, GPT-5.4, Qwen3.5-Plus, and DeepSeek-V3.2, reported alongside the full system and ablations in Tables 3 and 4. Averaged across these three, baseline Pass@1 drops sharply with difficulty ($0.48/0.23/0.08$ on easy / medium / hard). A broader comparison with PCBSchemaGen [27], CircuitLM [26], and GNN-based PCB schematic optimization [10] is in Appendix A.9.

Multi-Agent baseline. We additionally evaluate a multi-agent baseline on GPT-5.4 and Qwen3.5-Plus (Multi in Tables 3 and 4). Dedicated planner, generator, and checker agents coordinate task planning, schematic generation, and iterative review and repair, allowing at most six generation/repair attempts per trial. Its planner uses an objective and prompt distinct from TopoScope's module planner. This provides a high-capacity reference architecture, with additional computational cost from repeated generation and checking calls.

Results. Table 2 reports TopoScope across all seven models. Performance decreases with task complexity, but the drop is gradual rather than abrupt: averaged across models, Pass@1 is 0.89 on easy, 0.61 on medium, and 0.46 on hard designs, while APR remains high at 0.94, 0.90, and 0.89. For the three models with baseline runs, TopoScope improves Pass@1 on every tier, with the largest gains on medium and hard designs where single-pass generation is near its floor. Gemini-3.1-Pro is strongest overall, followed by GPT-5.4 and GLM-5. GPT-OSS-120B is competitive on easy designs (1.00 Pass@1), but its Pass@1 drops to 0.41 on medium and 0.27 on hard designs. This suggests that mid-scale locally hosted models can support simpler schematics, but still lag the best closed models on multi-IC designs. The relatively flat APR curve indicates that many hard-tier failures are near-misses under the calibrated threshold, not complete failures of basic connectivity.

4.3 Ablation Study

We isolate the contribution of TCS and the in-loop pattern-based ERC verification by removing each component while keeping the rest of the framework intact. The ablations are run on GPT-5.4 and Qwen3.5-Plus, the two models with the largest baseline-to-full gain in the main results. Tables 3 and 4 report Pass@1, Pass@3, and APR for the full system (Full), the multi-agent baseline (Multi), the no-TCS variant ($-$TCS, in which the model receives a flat context containing all components instead of a topology-scoped view), the no-ERC-loop variant ($-$ERC, in which generation is single-pass without in-loop pattern-based feedback), and the single-pass refined-knowledge baseline (Base).

Table 3: GPT-5.4 tier-level results. Pass@1, Pass@3, and APR (%) are macro-averaged across tasks within each tier. Bold denotes the highest score per tier and metric.

Tier	Pass@1					Pass@3					APR				
	Full	Multi	−TCS	−ERC	Base	Full	Multi	−TCS	−ERC	Base	Full	Multi	−TCS	−ERC	Base
Easy	**96.4**	85.7	64.3	82.1	50.0	**100.0**	99.3	74.3	99.3	50.0	**96.0**	93.8	91.8	94.1	88.2
Medium	**77.1**	68.6	41.4	48.6	34.3	**94.3**	83.1	48.9	83.1	48.0	**92.5**	91.4	88.6	87.5	86.9
Hard	**73.8**	52.4	50.0	42.9	14.3	**94.8**	69.0	70.0	71.4	38.1	**93.8**	88.6	88.5	90.6	83.8

Table 4: Qwen3.5-Plus tier-level results. Pass@1, Pass@3, and APR (%) are macro-averaged across tasks within each tier. Bold denotes the highest score per tier and metric.

Tier	Pass@1					Pass@3					APR				
	Full	Multi	−TCS	−ERC	Base	Full	Multi	−TCS	−ERC	Base	Full	Multi	−TCS	−ERC	Base
Easy	**96.4**	75.0	85.7	78.6	71.4	**100.0**	96.4	99.3	92.9	92.1	**95.8**	91.5	90.4	93.0	88.9
Medium	**72.9**	47.1	47.1	68.6	34.3	**91.1**	58.6	68.0	88.6	60.3	**92.4**	89.2	83.0	91.0	73.0
Hard	**46.7**	38.1	42.9	28.6	9.5	**72.2**	63.8	57.1	57.6	26.2	**89.7**	81.9	82.0	84.3	66.3

Full-system hallucination rate (HR), together with confidence intervals for the full system and Multi, is reported in Appendix A.5. HR is defined in Appendix A.10.

Effect of TCS and the pattern-based ERC loop. Removing TCS causes the largest single-condition drops in Pass@1, most consistently on medium designs (0.36 on GPT-5.4, 0.26 on Qwen3.5-Plus), with APR dropping in parallel; the failures it introduces are spread across multiple assertions per run rather than concentrated as near-misses. Removing the ERC loop, in contrast, scales monotonically with difficulty for GPT-5.4 ($0.14/0.29/0.31$ across easy / medium / hard), matching the prediction in Section 3.3 that the loop's role shifts from cleanup to correction as designs grow more complex; on Qwen3.5-Plus the effect is smaller and less monotonic, suggesting fewer errors are loop-recoverable. The hallucination rates reported in Appendix A.5 confirm that the two mechanisms address complementary error modes: TCS shapes what the model sees and prevents topology-level mistakes that the loop cannot easily recover from, while the ERC loop catches lower-level reference errors that survive a well-scoped context.

Comparison with the Multi-Agent baseline. The full system achieves higher observed Pass@1, Pass@3, and APR than Multi on every tier for both models. On medium and hard designs, the Pass@1 gaps are 8.5 and 21.4 percentage points for GPT-5.4, and 25.8 and 8.6 points for Qwen3.5-Plus. Despite explicit planning and iterative review and repair, Multi remains below TopoScope's graph-scoped generation on these point estimates. The confidence intervals in Appendix A.5 characterize sampling uncertainty; we do not infer statistical significance from the table alone.

5 Conclusion

We presented TopoScope, a training-free LLM agent for PCB schematic design. By coupling Topology-Guided Context Scoping with pattern-based rule verification over a datasheet-derived component knowledge base, TopoScope localizes the model's field of view at each generation step and catches chip-level errors as they appear. On our 20-design benchmark with assertion-based evaluation, TopoScope yields consistent gains across seven LLMs, with the largest improvements on harder multi-IC designs. By treating the schematic graph as a unifying substrate for generation, verification, and evaluation, this work bridges LLM reasoning and the pin-level reality of hardware, laying a foundation for robust schematic-level automation.

Limitations. TopoScope still depends on accurate structured component records extracted from datasheets, and its assertion-based evaluation assumes conversion to our schematic-graph and Connection Set representation. The current verifier mainly checks topology-level correctness, so production-ready use still requires numerical, layout-aware, thermal, signal-integrity, and human engineering review. Appendix A.11 provides a more detailed discussion.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China under Grant 62603111, the Postdoctoral Fellowship Program of CPSF under Grant GZC20260972, the Natural Science Foundation of Sichuan Province under Grant 2026NSFSC0317, and the Fundamental Research Funds for the Central Universities under Grants ZYGX2026J020, ZYGX2026K034, and ZYGX2025K028.

References

[1] H. Amano *et al.*, "The 2018 gan power electronics roadmap," *Journal of Physics D: Applied Physics*, vol. 51, no. 16, 2018.

[2] J. Pan, G. Zhou, C.-C. Chang, I. Jacobson, J. Hu, and Y. Chen, "A survey of research in large language models for electronic design automation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 30, no. 3, 2025.

[3] W. Qiu, Y. Xiao, and R. Pan, "Hwe-bench: Can language models perform board-level schematic designs?" *arXiv*, 2026.

[4] Z. Dong, W. Cao, M. Zhang, D. Tao, Y. Chen, and X. Zhang, "Cktgnn: Circuit graph neural network for electronic design automation," in *11th International Conference on Learning Representations, ICLR 2023*, 2023.

[5] Y. Shi, Z. Tao, Y. Gao, T. Zhou, C. Chang, Y. Wang, B. Chen, G. Zhang, A. Liu, Z. Yu, T.-J. Lin, and L. He, "Amsnet-kg: A netlist dataset for llm-based ams circuit auto-design using knowledge graph rag," *ACM Transactions on Design Automation of Electronic Systems*, vol. 30, no. 6, 2025.

[6] J. Gao, W. Cao, J. Yang, and X. Zhang, "Analoggenie: A generative engine for automatic discovery of analog circuit topologies," in *13th International Conference on Learning Representations, ICLR 2025*, 2025, pp. 48 314–48 332.

[7] C.-C. Chang, Y. Shen, S. Fan, J. Li, S. Zhang, N. Cao, Y. Chen, and X. Zhang, "Lamagic: Language-model-based topology generation for analog integrated circuits," in *Proceedings of Machine Learning Research*, vol. 235, 2024, pp. 6253–6262.

[8] R. Takagi, T. Masubuchi, Y. Moriguchi, and N. Takai, "Two-stage multi-objective bayesian optimization framework for analog circuit sizing," in *2025 International Symposium on Intelligent Signal Processing and Communication Systems, ISPACS 2025*, 2025.

[9] H. Li, J. Zhang, N. Xu, and M. Liu, "Fanoutnet: A neuralized pcb fanout automation method using deep reinforcement learning," in *Proceedings of the 37th AAAI Conference on Artificial Intelligence, AAAI 2023*, vol. 37, 2023, pp. 8554–8561.

[10] P. Plettenberg, A. Alcalde, B. Sick, and J. M. Thomas, "Graph neural networks for automatic addition of optimizing components in printed circuit board schematics," in *Lecture Notes in Computer Science*, vol. 16020 LNCS, 2026, pp. 508–524.

[11] P. Jansen, "From words to wires: Generating functioning electronic devices from natural language descriptions," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2023, pp. 12 972–12 990.

[12] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-chat: Challenges and opportunities in conversational hardware design," in *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD, MLCAD 2023*, 2023.

[13] S. Thakur, B. Ahmad, H. Pearce, B. Tan, B. Dolan-Gavitt, R. Karri, and S. Garg, "Verigen: A large language model for verilog code generation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 3, 2024.

[14] Y. Liu, C. Xu, Y. Zhou, Z. Li, and Q. Xu, "Deeprtl: Bridging verilog understanding and generation with a unified representation model," in *13th International Conference on Learning Representations, ICLR 2025*, 2025, pp. 3948–3970.

[15] M. Liu *et al.*, "Chipnemo: Domain-adapted llms for chip design," *arXiv*, 2023.

[16] H. Wu, Z. He, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu, "Chateda: A large language model powered autonomous agent for eda," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 10, pp. 3184–3197, 2024.

[17] Y. Fu, Y. Zhang, Z. Yu, S. Li, Z. Ye, C. Li, C. Wan, and Y. C. Lin, "Gpt4aigchip: Towards next-generation ai accelerator design automation via large language models," in *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, 2023.

[18] S. Thakur, J. Blocklove, H. Pearce, B. Tan, S. Garg, and R. Karri, "Autochip: Automating hdl generation using llm feedback," *arXiv*, 2023.

[19] K. Ye, Q. Yang, Z. Lu, H. Yu, T. Cui, R. Bai, and L. Shen, "Llm4netlist: Llm-enabled step-based netlist generation from natural language description," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 15, no. 2, pp. 337–348, 2025.

[20] Y. Lai, S. Lee, G. Chen, S. Poddar, M. Hu, D. Z. Pan, and P. Luo, "Analogcoder: Analog circuit design via training-free code generation," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 39, no. 1, 2025, pp. 379–387.

[21] P.-H. Chen, Y.-S. Lin, W.-C. Lee, T.-Y. Leu, P.-H. Hsu, A. Dissanayake, S. Oh, and C.-S. Chiu, "Menter: A fully-automated multi-agent workflow for end-to-end rf/analog circuits netlist design," in *Proceedings - 2025 IEEE International Conference on LLM-Aided Design, ICLAD 2025*, 2025, pp. 124–132.

[22] C. Liu and D. Chitnis, "Eeschematic: Multimodal-llm based ai agent for schematic generation of analog circuit," *arXiv*, 2025.

[23] J. Bhandari, V. Bhat, Y. He, H. Rahmani, S. Garg, and R. Karri, "Masala-chai: A large-scale spice netlist dataset for analog circuits by harnessing ai," *arXiv*, 2024.

[24] B. Liu, H. Zhang, X. Gao, Z. Kong, X. Tang, Y. Lin, R. Wang, and R. Huang, "Layoutcopilot: An llm-powered multiagent collaborative framework for interactive analog layout design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 44, no. 8, pp. 3126–3139, 2025.

[25] P. Vijayaraghavan, L. Shi, E. Degan, V. Mukherjee, and X. Zhang, "Autocircuit-rl: Reinforcement learning-driven llm for automated circuit topology generation," in *Proceedings of Machine Learning Research*, vol. 267, 2025, pp. 61 498–61 512.

[26] K. S. A. Hasan, S. R. Raiyan, H. M. Alvee, and W. Sadik, "Circuitlm: A multi-agent llm-aided design framework for generating circuit schematics from natural language prompts," *arXiv*, 2026.

[27] H. Zou, P. Han, E. Nazerian, and A. Q. Huang, "Pcbschemagen: Constraint-guided schematic design via llm for printed circuit boards (pcb)," *arXiv*, 2026.

[28] R. Matsuo, S. Uhlich, A. Venkitaraman, A. Bonetti, C.-Y. Hsieh, A. Momeni, L. Mauch, A. Capone, E. Ohbuchi, and L. Servadei, "Schemato - an llm for netlist-to-schematic conversion," in *2025 ACM/IEEE 7th Symposium on Machine Learning for CAD, MLCAD 2025*, 2025.

[29] OpenAI, "GPT-5.4 Thinking System Card," OpenAI, Tech. Rep., 2026, available: https://openai.com/index/gpt-5-4-thinking-system-card/. Accessed: Apr. 25, 2026.

[30] Google, "Gemini 3.1 Pro Preview," Google AI for Developers, 2026, available: https://ai.google.dev/gemini-api/docs/models/gemini-3.1-pro-preview. Accessed: Apr. 25, 2026.

[31] Qwen Team, "Qwen3.5: Towards Native Multimodal Agents," Qwen Blog, 2026, available: https://qwen.ai/blog?id=qwen3.5. Accessed: Apr. 25, 2026.

[32] GLM-5 Team, "GLM-5: From Vibe Coding to Agentic Engineering," *arXiv*, 2026.

[33] Kimi Team, "Kimi K2.5: Visual Agentic Intelligence," *arXiv*, 2026.

[34] DeepSeek-AI, "DeepSeek-V3.2: Pushing the Frontier of Open Large Language Models," *arXiv*, 2025.

[35] OpenAI, "gpt-oss-120b & gpt-oss-20b Model Card," *arXiv*, 2025.

[36] M. Chen *et al.*, "Evaluating large language models trained on code," *arXiv*, 2021.

[37] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, vol. 20, pp. 37–46, 1960.

A Appendix

Benchmark and evaluation. The 20 per-design specifications are listed in A.1, and the functional assertion types together with per-design enumerations are catalogued in A.4. Confidence intervals for the overall results in Table 2 and the full-system results in Tables 3 and 4 are reported in A.5.

Structured inputs. The unified JSON schemas for component descriptions and reference circuits are documented in A.2, and the full pattern-based ERC rule set used during in-loop verification is given in A.3.

Prompts. The full templates used for macro planning, per-component wiring, and the baseline are provided in A.6.

Results and analysis. Assertion-level breakdowns beyond the aggregate metrics of Section 4 are reported in A.7. The Cohen's κ expert study and APR threshold calibration are described in A.8. A detailed comparison with PCBSchemaGen, CircuitLM, and GNN-based PCB schematic optimization is in A.9, and the hallucination rate is formalized together with a per-mechanism breakdown in A.10.

Limitations. An extended limitations discussion is in A.11.

A.1 Benchmark Dataset Details

The 20 designs in our benchmark are derived from manufacturer typical-application circuits, simplified for tractable end-to-end evaluation while preserving the key topological constraints of each domain. Designs are stratified into three difficulty tiers based on the number of integrated circuits and the topological complexity of inter-component connectivity. Easy contains four designs with one or two integrated circuits, each exercising a single core function such as LDO regulation, USB-to-UART bridging, audio amplification, or constant-current LED driving. Medium contains ten four-IC designs that introduce shared buses, multi-rail regulation, and analog-digital interfacing. Hard contains six designs of five to seven integrated circuits, combining multiple of these mechanisms within a single board.

Each task specifies the exact part numbers, input and output voltage rails, bus topology (shared I2C bus, dedicated SPI lines), grounding convention (common ground, isolated ground), and constraints on unused channels or pins. The model generates the connectivity, supporting passives, and net names. The prompt templates are given in A.6.

Table 5 lists all 20 designs with their domain type, condensed task description, and the integrated circuits involved.

A.2 Structured Knowledge Schemas

The component knowledge base of TopoScope stores per-component information as JSON records, one record per part number. Each record is the unit of retrieval during topology-guided context scoping: when the model wires a component, the corresponding record is fetched in full and inserted into the scoped context, together with any reference circuits attached to it. The schema has five fields, summarized below using representative fragments from an SN74HC595 record.

`part_number`. A unique string identifier serving as the retrieval key.

```
"part_number": "SN74HC595"
```

`description`. A short natural-language description providing the model with auxiliary context about what the component is and what it does.

```
"description": "8-Bit Shift Register with 3-State Output Registers, Serial Input/Parallel Output"
```

`pins`. A list of pin entries, one per physical pin. Each entry carries a number, a name, an electrical type, and a free-form description. The electrical type is drawn from a fixed enumeration: Input, Output, Bidirectional, PowerIn, PowerOut, Ground, Passive. These entries become the pin-level nodes of the schematic graph defined in Section 3.

```
"pins": [
  {"number": "8", "name": "GND", "electrical_type": "Ground", "description": "Ground pin"},
```

Table 5: Detailed Benchmark designs.

ID	Type	Design Task Description	Key Components
1	COM	USB-to-UART interface using CH340 to bridge USB to TXD/RXD; XC6206 derives 3.3V from the USB 5V input.	CH340, XC6206
2	PM	Regulated 3.3V output stage from a 5V input using AMS1117-3.3, monitored by TPS3703 producing a POWER_GOOD signal.	AMS1117-3.3, TPS3703
3	LED	12-24V to 350mA constant-current driver for a 1W LED, using PT4115.	PT4115
4	AUD	Single 9V supply two-stage audio amplifier driving an 8-ohm speaker; NE5532 preamp followed by LM386 power stage, total gain 100x-200x.	NE5532, LM386
5	SEN	STM32F103 reading BME280 and driving SSD1306 OLED on a shared I2C bus; AMS1117-3.3 provides 3.3V from a 5V input.	AMS1117-3.3, STM32F103, BME280, SSD1306
6	SEN	ATmega328P reading MPU-6050 over I2C and logging to W25Q64JV over SPI; ME6211 provides 3.3V from a 5V input.	ME6211, ATmega328P, MPU-6050, W25Q64JV
7	COM	USB development board with CH32V003 RISC-V MCU, CH340 USB-to-UART bridge, and AT24C256 EEPROM; XC6206 derives 3.3V from USB 5V.	XC6206, CH340, CH32V003, AT24C256
8	COM	STM32F103 to MCP2515 standalone CAN controller over SPI, driving SN65HVD230 differential transceiver; AMS1117-3.3 from 5V.	AMS1117-3.3, STM32F103, MCP2515, SN65HVD230
9	COM	GD32F103 to SP3485 RS485 transceiver over UART, with W25Q128JV Flash logging over SPI; AMS1117-3.3 from 5V.	AMS1117-3.3, GD32F103, SP3485, W25Q128JV
10	PM	MCU-less three-stage cascade 12V to 5V to 3.3V to 1.8V; one LM393 channel monitors 3.3V for POWER_GOOD, the other channel safely tied off.	TPS54331, AMS1117-3.3, ME6211, LM393
11	DRV	STM32F103 driving TB6612FNG channel A; INA128 senses a low-side shunt for motor current; common GND with separate motor and 5V rails.	AMS1117-3.3, STM32F103, TB6612FNG, INA128
12	AUD	ATmega328P controlling MCP4725 over I2C; MCP4725 VOUT biases the NE5532 channel-1 non-inverting input; channel 2 left in a safe state.	AMS1117-3.3, ATmega328P, MCP4725, NE5532
13	LED	STM32F103 driving MAX7219 LED matrix driver over SPI, cascaded to 74HC595 shift register; both 5V-supplied.	AMS1117-3.3, STM32F103, MAX7219, 74HC595
14	SEN	STM32F103 reading ADS1115 over I2C; ADS1115 measures one external analog input and monitors REF5050 as a precision reference.	AMS1117-3.3, STM32F103, ADS1115, REF5050
15	SEN	Flight controller core with STM32F427, separate digital 3.3V (TPS54331) and analog 3.3V (ME6211) rails, MPU-6050 over I2C, BMP280 and W25Q128JV on shared SPI.	TPS54331, ME6211, STM32F427, MPU-6050, BMP280, W25Q128JV
16	COM	USB-C-powered IoT gateway: TP4056 charging, DW01A protection, ME6211 to 3.3V; ESP32-WROOM-32E reads BME280 over I2C; CP2102N debug UART.	CP2102N, TP4056, DW01A, ME6211, ESP32-WROOM-32E, BME280
17	DRV	STM32F103 controlling two TMC2209 stepper drivers over UART (distinct addresses) plus W25Q64JV over SPI; 12-24V to 5V via TPS54331, then to 3.3V via AMS1117-3.3.	TPS54331, AMS1117-3.3, STM32F103, TMC2209 ($\times$2), W25Q64JV
18	AUD	Stereo audio chain: ATmega328P-controlled MCP4725 biases stereo NE5532 preamp into PAM8403 class-D output stage; MT3608 boosts USB 5V to 9V for the preamp only.	MT3608, AMS1117-3.3, ATmega328P, MCP4725, NE5532, PAM8403
19	PM	MCU-less branched supply: TPS54331 5V from 12V, TPS61088 boosts to 12V load rail, AMS1117-3.3 to 3.3V, ME6211 to 1.8V; TPS3703 monitors 5V.	TPS54331, TPS61088, AMS1117-3.3, ME6211, TPS3703
20	SEN	Galvanically isolated ADC node: STM32F103 with ADS1118 across two ISO7721 isolators on SPI; REF5050 as isolated-side reference; GND and ISO_GND never join.	XC6206 ($\times$2), STM32F103, ISO7721 ($\times$2), ADS1118, REF5050

```
    {"number": "16", "name": "VCC", "electrical_type": "PowerIn", "description": "Power supply pin"},
    {"number": "14", "name": "SER", "electrical_type": "Input", "description": "Serial data input"}
  ]
```

`erc_rules`. A list of pattern-based electrical rule check entries attached to this component. Each entry has a rule type drawn from a fixed enumeration (full set in A.3), a severity level (warning or error), and a parameter block specifying target pins and supporting context. Two examples are shown below: a 0.1uF bypass capacitor required between VCC and ground, and a forbid-floating constraint on all unused inputs.

```
"erc_rules": [
  {"rule_type": "REQUIRE_BYPASS_CAP",
   "severity": "warning",
   "parameters": {"target_pins": ["VCC"], "reference_pin_type": "Ground", "cap_values": ["0.1uF"]}},
  {"rule_type": "FORBID_FLOATING",
   "severity": "error",
   "parameters": {"target_pins": ["OE", "SER", "SRCLK", "RCLK", "SRCLR"], "active_condition": "Always"}}
]
```

`reference_circuits`. A list of datasheet-derived typical-application subgraphs attached to the component. Each entry describes a reusable sub-circuit through two fields. The required-passives field lists the supporting passive components, including their reference designators, types, values, and roles. The topology-nets field enumerates the nets that wire the component's pins together with these passives. These entries serve as graph-level grounding patterns: when TopoScope wires a component, the matching reference circuit is included in the scoped context as a concrete example of the expected topology.

```
"reference_circuits": [
  {"circuit_name": "Basic Serial-to-Parallel Shift Register Configuration",
   "required_passives": [
     {"ref": "C_bypass", "type": "Capacitor", "value": "0.1uF", "role": "VCC bypass capacitor"}],
   "topology_nets": [
```

Table 6: Chip-level pattern classes used by the in-loop ERC verifier. These are the two process ERC classes from Section 3.3. System-level patterns are evaluated externally as functional assertions when they express board-level intent.

Pattern class	Graph scope	When checked	Representative rules
Adjacency patterns	One- or two-hop neighborhood around an anchor pin	During component generation	`REQUIRE_BYPASS_CAP`, `REQUIRE_PULL_UP_DOWN`, `FORBID_FLOATING`, `MANDATORY_NET_CONNECTION`, `OPEN_DRAIN_OUTPUT_PULLUP`
Path patterns	Bounded local path or small passive topology between named anchors	During component generation	`REQUIRE_FEEDBACK_NETWORK`, `REQUIRE_SENSE_RESISTOR`, `REQUIRE_OSC_LOAD_CAPS`, `REQUIRE_GAIN_SETTING_NETWORK`, `REQUIRE_TIMING_NETWORK`

```
    {"name": "VCC", "nodes": [{"pin": "VCC"}, {"ref": "C_bypass", "pin": "1"}]},
    {"name": "GND", "nodes": [{"pin": "GND"}, {"ref": "C_bypass", "pin": "2"}]}]}
]
```

A.3 Pattern-based ERC Rules

ERC rules are datasheet-derived chip-level constraints attached to component records in the structured knowledge base. The component-generation loop checks their adjacency and path patterns on each proposed connection set. System-level constraints, such as voltage-domain consistency and ground separation, are evaluated on the completed graph through the functional assertions in A.4.

Each ERC entry has a uniform rule-object format. The field `rule_type` selects a rule from the fixed vocabulary, `severity` records whether the violation should be treated as an error or a warning, and `parameters` provides the pins, net classes, values, and local conditions needed by the matcher. A typical bypass-capacitor rule is shown below.

```
{
  "rule_type": "REQUIRE_BYPASS_CAP",
  "severity": "warning",
  "parameters": {
    "target_pins": ["VCC"],
    "reference_pin_type": "Ground",
    "cap_values": ["0.1uF"]
  }
}
```

Formally, an ERC rule instance is a tuple

$$r = (t, s, \theta, \beta), \tag{7}$$

where t is the rule type, s is the severity level, θ is the parameter object, and β binds the named anchors in the pattern template to concrete schematic pins. The rule type maps to a graph pattern template P_t, and the parameter object instantiates it as $P_t(\theta)$. Let $\Phi(G, P)$ denote the set of injective mappings that satisfy Definition 2 for pattern P in graph G. Given a schematic graph G, verification returns

$$\mathrm{ERC}(G, r) = \mathbb{1}\left[\exists \phi \in \Phi(G, P_t(\theta)) : \phi|_{\mathrm{dom}(\beta)} = \beta\right]. \tag{8}$$

The mapping ϕ follows the label-constrained satisfaction semantics in Definition 2. ERC parameters can bind multiple named anchors, such as the two oscillator pins of a crystal rule. Severity s controls failure reporting and downstream filtering, independently of the matching semantics.

Process ERC pattern classes. The in-loop ERC verifier uses two chip-level pattern classes: adjacency patterns and path patterns. Table 6 summarizes the distinction. Both classes inspect a local region around the component being generated, which makes them suitable for immediate feedback within the scoped view.

Adjacency patterns. Adjacency patterns check whether a required local neighbor exists, or whether an immediately illegal local condition is absent. They usually inspect the target pin, the net attached to it, the other pins on that net, and the components owning those pins. For example, `REQUIRE_BYPASS_CAP` anchors on a power pin and searches for a capacitor whose other side reaches ground. `OPEN_DRAIN_OUTPUT_PULLUP` anchors on an open-drain output and searches for a pull-up path to a power net. `FORBID_FLOATING` is the complementary case, where the matcher verifies that a required pin has at least one legal attachment.

Table 7: ERC rule vocabulary used by the in-loop component-level verifier. Class labels refer to Table 6. Board-level voltage, ground-isolation, and communication-pair constraints are specified as functional assertions in A.4.

Rule type	Class	Purpose
REQUIRE_BYPASS_CAP	Adj.	Require a bypass or decoupling capacitor from a target pin or rail to a reference net, usually ground.
REQUIRE_PULL_UP_DOWN	Adj.	Require a pull-up or pull-down resistor for configuration pins, bus lines, or reset pins.
FORBID_FLOATING	Adj.	Require specified pins to be connected under the stated active condition.
MANDATORY_NET_CONNECTION	Adj.	Require specified pins to connect to a required net class such as power or ground.
FORBIDDEN_CONNECTION	Adj.	Forbid DNC, NC, or internal pins from being connected.
OPEN_DRAIN_OUTPUT_PULLUP	Adj.	Require an open-drain or open-collector output to have a pull-up path.
REQUIRE_PARALLEL_PROTECTION	Adj.	Require ESD, TVS, or surge protection from exposed interface pins to a reference net.
REQUIRE_FEEDBACK_NETWORK	Path	Require a regulator feedback divider between the output rail, feedback pin, and ground.
REQUIRE_SERIES_PASSIVE	Path	Require a resistor, inductor, or ferrite bead in series with a target signal or supply path.
REQUIRE_SENSE_RESISTOR	Path	Require a current-sense resistor topology near sensing or driver pins.
REQUIRE_GAIN_SETTING_NETWORK	Path	Require an op-amp or instrumentation-amplifier gain-setting network.
REQUIRE_TIMING_NETWORK	Path	Require an RC timing network for timing, soft-start, oscillator, or PWM-control pins.
REQUIRE_RESONATOR	Path	Require an external resonator or crystal between oscillator pins, optionally with load capacitors.
REQUIRE_GATE_BASE_RESISTOR	Path	Require a gate or base resistor, optionally with a pull-down element, on transistor-control paths.
REQUIRE_FREEWHEELING_DIODE	Path	Require a freewheeling diode path for inductive loads or switching nodes.
REQUIRE_CURRENT_LIMIT_RESISTOR	Path	Require a current-limiting resistor for LEDs, optocouplers, or similar devices.
REQUIRE_AUDIO_STABILITY_NETWORK	Path	Require an audio stability network such as a Zobel network or AC-coupling path.
REQUIRE_OSC_LOAD_CAPS	Path	Require load capacitors from both oscillator-side nets to ground.
REQUIRE_PROGRAMMING_RESISTOR	Path	Require a programming resistor or related local configuration network.

Path patterns. Path patterns require a small topology rather than a single neighbor. These rules use a bounded traversal with component-type constraints on the intermediate elements. For a switching regulator, REQUIRE_FEEDBACK_NETWORK checks that the feedback pin participates in a divider between the output rail and ground. For a crystal oscillator, REQUIRE_OSC_LOAD_CAPS checks that the two oscillator pins are on distinct nets and that each side has a load capacitor to ground. For amplifier and sensing circuits, REQUIRE_GAIN_SETTING_NETWORK and REQUIRE_SENSE_RESISTOR detect gain-setting and measurement structures in the local subgraph.

Complete in-loop rule vocabulary. Table 7 lists the ERC rule types used by the in-loop verifier. The same rule type may be assigned different parameters for different components. For example, a bypass rule for an MCU power pin and a bypass rule for an LDO input pin share the same matcher, but the target pin list and expected capacitor values come from different datasheets.

Use in the generation loop. When TopoScope processes a component, it retrieves the component's ERC entries from the structured knowledge schema in A.2. The adjacency and path entries are checked on the tentative graph obtained after the model proposes that component's connection set. Failed rules are converted into feedback and the component is regenerated within the same scoped view. This schedule keeps most violations local to the component that caused them. Board-level constraints that require a fully merged graph, including voltage-domain propagation, ground isolation, and pair-level communication constraints, are handled by the functional assertion layer rather than by the process ERC loop.

A.4 Functional Assertion Specification

Functional assertions are the external evaluation layer for the benchmark. As contrasted with ERC rules in A.3, they are attached to a benchmark design and are evaluated only after the full schematic graph has been generated and virtual-port references have been resolved. They encode the intended board-level behavior of the task, such as whether all devices on an I2C bus share the same two nets, whether analog and digital rails remain separated, whether a feedback network exists, and whether a multi-stage communication chain is electrically reachable.

Table 8: Functional assertion classes used for post-generation evaluation. The classes are design-level predicates evaluated on the completed schematic graph.

Class	Representative checks	Design intent captured
Pin-state checks	`PIN_NOT_FLOATING`, `MUST_BE_FLOATING`	Require a specific functional pin to be connected, or require a pin marked NC or DNC to remain unconnected. Global single-net validity is handled as a precheck.
Connectivity checks	`SAME_NET`, `CONNECTED_TO_GROUND`, `CONNECTED_TO_POWER`, `CONNECTED_TO`, `CONNECTED_TO_NET_CLASS`	Verify direct net sharing, power attachment, ground attachment, and required rail membership. These checks capture shared buses and common power domains.
Isolation checks	`NOT_SAME_NET`, `NET_GROUP_ISOLATION`, `PIN_NOT_ON_NET_WITH`	Prevent shorts across incompatible nets, voltage domains, bus lines, and ground regions. These checks are used for rail separation, SDA/SCL separation, and isolation barriers.
Topology checks	`HAS_CAP_TO_GROUND`, `HAS_CRYSTAL_PAIR`, `HAS_PULLUP_TO_POWER`, `HAS_FEEDBACK_DIVIDER`, `HAS_BUS_TERMINATION`	Require small supporting subgraphs such as decoupling, oscillator loading, pull-ups, feedback dividers, current-sense elements, freewheeling paths, and termination resistors.
Signal-chain checks	`EXISTS_PATH`, `SIGNAL_CHAIN_ORDER`	Verify reachability through an intended multi-stage chain while constraining which component types may be traversed. These checks express relations such as MCU to CAN controller to transceiver paths.
Compositional checks	`ANY_OF`, `ALL_OF`	Express legal alternatives and grouped requirements without duplicating assertions.

Each benchmark design d is associated with an assertion set $\mathcal{A}_d = \{a_1, \ldots, a_m\}$. Each assertion is a typed predicate over the generated schematic graph. We write an assertion as

$$a = (q, \theta, \gamma), \tag{9}$$

where q is the assertion type, θ is the parameter object, and γ is an optional composition operator. Atomic assertions are evaluated by a type-specific matcher,

$$F_a(G) = f_q(G; \theta) \in \{0, 1\}. \tag{10}$$

Composite assertions combine child predicates with disjunction or conjunction. We use them only when a design has multiple legitimate implementations, such as alternative MCU pins for the same bus function or alternative enable-pin configurations. A disjunction passes when at least one child predicate passes, while a conjunction passes only when all child predicates pass.

The reported Assertion Pass Rate (APR) for a generated graph G on design d is

$$\mathrm{APR}(G, d) = \frac{1}{|\mathcal{A}_d|} \sum_{a \in \mathcal{A}_d} F_a(G). \tag{11}$$

Structural validity is checked before APR is interpreted as a pass/fail signal. A pin assigned to more than one net invalidates the run. These global prechecks are excluded from the functional assertion count.

Assertion object format. In the benchmark files, each assertion is stored as a compact JSON object with a stable identifier, a check type, a short natural-language description, and a parameter object. Composite assertions add a composition type and a list of child conditions. The implementation also accepts a few compatibility aliases for older files, but the logical form is the same.

```
{
  "id": "T2-1-I2C-SDA",
  "check": "SAME_NET",
  "desc": "MCU, BME280, and SSD1306 share the I2C SDA line.",
  "params": {
    "pins": ["U2.PB7", "U3.SDA", "U4.SDA"]
  }
}
```

Each assertion is evaluated deterministically on the generated Connection Set or schematic graph.

Assertion classes. Table 8 groups assertions by the graph property being checked. Candidate pins and candidate net groups represent alternative legal pin mappings for the same intended topology.

Connectivity and isolation semantics. Connectivity assertions operate on the net equivalence classes induced by the generated Connection Set. A SAME_NET assertion passes when all specified pins, or at least one candidate from each specified pin group, are members of the same net.

CONNECTED_TO_GROUND and CONNECTED_TO_POWER check whether the target pin's net contains a compatible ground or power source. Isolation assertions are the dual of these checks. NOT_SAME_NET rejects a short between two pins, while NET_GROUP_ISOLATION rejects any net overlap between two or more incompatible groups. These predicates are especially important for differentiating valid shared buses from harmful shorts. For example, all I2C SDA pins should share one net, but SDA and SCL must remain distinct.

Topology and signal-chain semantics. Topology assertions match small supporting structures around an anchor pin or a pair of pins. HAS_CAP_TO_GROUND searches for a capacitor path from the target net to ground. HAS_CRYSTAL_PAIR checks that the two oscillator pins occupy distinct nets and that both nets have load capacitors to ground. HAS_FEEDBACK_DIVIDER checks that a feedback pin participates in a divider between an output rail and ground. These checks are stricter than direct connectivity, but still remain local enough to be interpreted as intended subgraphs.

Signal-chain assertions cover cases where correctness is not a single shared net. EXISTS_PATH checks whether a source pin reaches a destination pin or net within a bounded number of hops, optionally restricting traversal to specified component classes. SIGNAL_CHAIN_ORDER applies the same idea to a sequence of endpoints. These assertions are used sparingly. Whenever SAME_NET already implies the desired relation, we avoid adding a redundant path assertion.

Non-redundancy principle. Assertions are designed so that each item contributes independent information to APR. If one assertion logically implies another, the implied assertion is removed. For example, a SAME_NET assertion over an I2C line already implies pairwise reachability among the listed pins, so an additional EXISTS_PATH assertion over the same endpoints would be redundant. Similarly, once all ground pins are required to share one net, a single ground-property check is sufficient. This principle keeps APR interpretable: a lower APR indicates missing functional intent rather than a large number of repeated checks over the same net.

Relation to Pass@k. For each run, APR measures the fraction of functional assertions that pass. A run is counted as a full pass only if it is structurally valid and the APR meets the calibrated threshold for the design tier, as described in A.8. APR measures functional completion, while Pass@k measures thresholded reliability over repeated samples.

A.5 Confidence Interval Report

Table 9 reports overall Pass@1 and Pass@3 across all 20 designs corresponding to Table 2. Tables 10 and 11 report tier-level Pass@1, Pass@3, APR, and HR for Full, Multi-agent, −TCS, −ERC, and Baseline, corresponding to Tables 3 and 4. Entries use $x^{+u}_{-\ell}$, where x is the point estimate and $u = U - x$, $\ell = x - L$ for 95% confidence limits L, U; values are percentages. Each design has seven runs, except the six hard designs evaluated with the full Qwen3.5-Plus system, which have ten after adding three samples per design. Missing scheduled runs are counted as failures (zero APR and unit HR), and duplicate run IDs are ignored.

Calculation. All aggregate intervals use a two-level percentile bootstrap with 20,000 replicates and random seed 2026. In each replicate, we sample designs with replacement, then sample the original number of runs with replacement within each selected design. We recompute the design-level metrics and take their unweighted macro-average. For Pass@k, the recomputed score is $1 - \binom{n-c}{k} / \binom{n}{k}$, where n and c are the resampled run count and pass count. Overall resampling is stratified by difficulty, preserving the four easy, ten medium, and six hard designs; tier-specific resampling stays within the corresponding tier. The interval endpoints are the 2.5th and 97.5th percentiles of the bootstrap aggregate scores. Individual-design Pass@1 intervals use Wilson's method; aggregate intervals use the bootstrap above.

Table 9: Overall full-system results corresponding to Table 2. Point estimates with upper/lower 95% CI errors (%). Each design receives equal weight.

Model	**Pass@1**	**Pass@3**
DeepSeek-V3.2	$30.7^{+11.4}_{-10.7}$	$51.7^{+13.4}_{-19.0}$
GPT-OSS-120B	$45.7^{+11.4}_{-10.0}$	$68.1^{+10.9}_{-17.2}$
Kimi-K2.5	$46.4^{+14.3}_{-14.3}$	$70.3^{+11.4}_{-19.3}$
Qwen3.5-Plus	$69.7^{+12.3}_{-13.1}$	$87.2^{+8.9}_{-14.7}$
GLM-5	$77.1^{+12.9}_{-13.5}$	$90.3^{+8.4}_{-14.0}$
GPT-5.4	$80.0^{+10.0}_{-11.4}$	$95.6^{+4.1}_{-11.0}$
Gemini-3.1-Pro	$85.0^{+10.0}_{-12.1}$	$95.1^{+4.8}_{-10.8}$

Table 10: GPT-5.4 tier-level results corresponding to Table 3. Point estimates with upper/lower 95% CI errors (%).

Tier	**Method**	**Pass@1**	**Pass@3**	**APR**	**HR**
Easy	Full	$96.4^{+3.6}_{-10.7}$	$100.0^{+0.0}_{-0.7}$	$96.0^{+3.2}_{-3.9}$	$3.6^{+10.7}_{-3.6}$
	Multi-agent	$85.7^{+14.3}_{-25.0}$	$99.3^{+0.7}_{-13.6}$	$93.8^{+6.2}_{-6.5}$	$10.7^{+25.0}_{-10.7}$
	−TCS	$64.3^{+35.7}_{-39.3}$	$74.3^{+25.7}_{-49.3}$	$91.8^{+8.2}_{-8.6}$	$0.0^{+0.0}_{-0.0}$
	−ERC	$82.1^{+17.9}_{-25.0}$	$99.3^{+0.7}_{-13.6}$	$94.1^{+4.3}_{-6.0}$	$21.4^{+28.6}_{-21.4}$
	Baseline	$50.0^{+50.0}_{-50.0}$	$50.0^{+50.0}_{-50.0}$	$88.2^{+11.8}_{-12.2}$	$25.0^{+50.0}_{-25.0}$
Medium	Full	$77.1^{+14.3}_{-18.5}$	$94.3^{+5.7}_{-17.4}$	$92.5^{+2.7}_{-2.6}$	$4.3^{+7.1}_{-4.3}$
	Multi-agent	$68.6^{+22.8}_{-24.3}$	$83.1^{+15.5}_{-25.7}$	$91.4^{+4.6}_{-5.7}$	$10.0^{+17.1}_{-10.0}$
	−TCS	$41.4^{+28.6}_{-27.1}$	$48.9^{+30.0}_{-29.2}$	$88.6^{+3.6}_{-3.7}$	$5.7^{+8.6}_{-5.7}$
	−ERC	$48.6^{+18.5}_{-18.6}$	$83.1^{+10.9}_{-25.7}$	$87.5^{+3.5}_{-3.7}$	$34.3^{+20.0}_{-18.6}$
	Baseline	$34.3^{+25.7}_{-22.9}$	$48.0^{+26.0}_{-29.4}$	$86.9^{+4.1}_{-4.7}$	$30.0^{+21.4}_{-18.6}$
Hard	Full	$73.8^{+19.1}_{-23.8}$	$94.8^{+5.2}_{-21.0}$	$93.8^{+3.4}_{-3.9}$	$16.7^{+16.6}_{-11.9}$
	Multi-agent	$52.4^{+33.3}_{-33.4}$	$69.0^{+28.6}_{-37.6}$	$88.6^{+9.1}_{-12.3}$	$35.7^{+33.3}_{-28.6}$
	−TCS	$50.0^{+33.3}_{-31.0}$	$70.0^{+25.7}_{-37.1}$	$88.5^{+6.6}_{-7.4}$	$23.8^{+31.0}_{-21.4}$
	−ERC	$42.9^{+28.5}_{-26.2}$	$71.4^{+21.5}_{-33.3}$	$90.6^{+2.9}_{-4.0}$	$54.8^{+30.9}_{-31.0}$
	Baseline	$14.3^{+14.3}_{-11.9}$	$38.1^{+24.8}_{-31.0}$	$83.8^{+7.6}_{-8.3}$	$83.3^{+11.9}_{-14.3}$

Table 11: Qwen3.5-Plus tier-level results corresponding to Table 4. Point estimates with upper/lower 95% CI errors (%).

Tier	Method	Pass@1	Pass@3	APR	HR
Easy	Full	$96.4^{+3.6}_{-10.7}$	$100.0^{+0.0}_{-0.7}$	$95.8^{+3.5}_{-4.7}$	$14.3^{+25.0}_{-14.3}$
	Multi-agent	$75.0^{+25.0}_{-28.6}$	$96.4^{+3.6}_{-22.1}$	$91.5^{+7.4}_{-7.3}$	$46.4^{+50.0}_{-46.4}$
	−TCS	$85.7^{+14.3}_{-21.4}$	$99.3^{+0.7}_{-13.6}$	$90.4^{+9.2}_{-12.9}$	$42.9^{+46.4}_{-42.9}$
	−ERC	$78.6^{+21.4}_{-35.7}$	$92.9^{+7.1}_{-28.6}$	$93.0^{+7.0}_{-8.6}$	$42.9^{+46.4}_{-42.9}$
	Baseline	$71.4^{+28.6}_{-35.7}$	$92.1^{+7.9}_{-31.4}$	$88.9^{+10.7}_{-13.5}$	$17.9^{+28.5}_{-17.9}$
Medium	Full	$72.9^{+18.5}_{-21.5}$	$91.1^{+8.6}_{-20.0}$	$92.4^{+3.0}_{-3.2}$	$4.3^{+8.6}_{-4.3}$
	Multi-agent	$47.1^{+27.2}_{-27.1}$	$58.6^{+25.7}_{-30.0}$	$89.2^{+4.7}_{-5.0}$	$18.6^{+21.4}_{-15.7}$
	−TCS	$47.1^{+22.9}_{-22.8}$	$68.0^{+21.4}_{-30.6}$	$83.0^{+7.2}_{-8.9}$	$14.3^{+17.1}_{-12.9}$
	−ERC	$68.6^{+17.1}_{-20.0}$	$88.6^{+11.1}_{-23.2}$	$91.0^{+3.5}_{-4.5}$	$15.7^{+17.2}_{-12.8}$
	Baseline	$34.3^{+21.4}_{-18.6}$	$60.3^{+20.0}_{-28.9}$	$73.0^{+12.5}_{-14.4}$	$27.1^{+15.8}_{-14.2}$
Hard	Full	$46.7^{+25.0}_{-26.7}$	$72.2^{+24.5}_{-35.5}$	$89.7^{+4.8}_{-5.5}$	$21.7^{+30.0}_{-20.0}$
	Multi-agent	$38.1^{+26.2}_{-26.2}$	$63.8^{+28.6}_{-40.0}$	$81.9^{+7.1}_{-10.3}$	$52.4^{+26.2}_{-26.2}$
	−TCS	$42.9^{+35.7}_{-33.4}$	$57.1^{+31.5}_{-40.4}$	$82.0^{+11.7}_{-14.5}$	$42.9^{+28.5}_{-26.2}$
	−ERC	$28.6^{+23.8}_{-19.1}$	$57.6^{+25.7}_{-36.2}$	$84.3^{+7.4}_{-10.7}$	$45.2^{+31.0}_{-30.9}$
	Baseline	$9.5^{+14.3}_{-9.5}$	$26.2^{+24.3}_{-26.2}$	$66.3^{+14.4}_{-16.2}$	$69.0^{+16.7}_{-19.0}$

A.6 Prompt Templates

The condensed prompt templates below cover shared design knowledge, global planning, scoped component wiring, and ERC-guided repair. At runtime, they are populated with full component records, graph-tool schemas, graph summaries, and a detailed naming policy.

Shared design knowledge. All TopoScope prompts begin with a shared block that states the electrical invariants and naming requirements. This block keeps the phase-specific prompts short and ensures that the same graph contract is used by planning, wiring, and repair.

Shared Design Knowledge

Role. You are an electronic design engineer performing schematic-level netlist generation. Given a system description, pre-selected ICs, component pin data, reference circuits, and module interfaces, produce the electrical connectivity for a working PCB schematic.

Electrical invariants.

- Pins connect to nets, never directly to other pins.
- Use only the provided ICs and the provided pin names. Do not invent pins or active components.
- Follow each IC's datasheet reference circuit when a template is provided.
- Instantiate required passives, including bypass capacitors, feedback dividers, crystal load capacitors, pull-ups, sense resistors, inductors, and diodes.
- Power pins and ground pins must never share a net.
- Do not connect two strong drivers to the same net.
- Critical pins must not float. This includes power inputs, grounds, enable pins, chip-enable pins, and unused flash WP or HOLD pins.

Naming requirements.

- Refdes uses standard numeric designators such as U1, R1, C1, L1, D1, Q1, Y1, F1, J1, and SW1.
- IC pin IDs use datasheet pin names, for example U1.VIN and U2.PB7.

- Passive part numbers use TYPE_VALUE notation, for example CAP_100nF, RES_10K, IND_4.7uH, and DIODE_SS34.
- Net names use UPPER_SNAKE_CASE and semantic names, such as VCC_3V3, I2C_SDA, SPI_CS_U3, U1_SW, and U2_OSC_IN.
- Module names use lower_snake_case and describe function rather than a specific IC.

Component data. Each IC record provides pins, reference circuits, and ERC rules. Use pins as the authoritative pin table, use reference circuits as wiring templates, and treat ERC rules as constraints that will be checked after wiring.

Phase 0: global planning. The global planning prompt constructs module boundaries and inter-module interfaces. It does not instantiate passives or emit pin-level connections. Its output is a plan that Phase 1 can expand under TCS.

Phase 0 Global Planning Prompt

Task. Create a global schematic plan for the requested PCB design.

Inputs.

- System functional description.
- Pre-selected ICs and short component descriptions.
- Board-level input and output nets.
- Summaries of available component knowledge records.

Output. Return a JSON object with two fields: modules and macro_connections. Each module lists its functional description, assigned components, and virtual ports. Each macro connection links two module ports and assigns the intended net name.

Rules.

- Plan module interfaces only. Do not instantiate passives and do not write pin-level connections.
- Do not introduce new active ICs.
- Global nets such as GND, VIN, VBAT, and externally specified rails are board-wide. Do not redeclare them as virtual ports.
- Internally generated rails must be exported by the provider module and imported by every consumer module.
- Shared buses reuse the same net name across all participating modules, for example I2C_SDA, I2C_SCL, SPI_MOSI, SPI_MISO, and SPI_CLK.
- Virtual ports are temporary module-boundary declarations. They are resolved into ordinary nets during graph merge.

Phase 1: scoped component wiring. The Phase 1 prompt is called once per focused component. The model receives the current component in full, the visible active nets, the module's virtual ports, global nets, and a folded summary of completed components. It may reuse visible nets, create local nets, instantiate passives, and connect pins through graph-operation tools. It may not reference entities outside the scoped view.

Phase 1 Scoped Wiring Prompt

Task. Wire the current component within the current module. Produce a complete local subgraph around this component, including required passives.

Inputs.

- Current component and its full pin table.
- Reference circuits and required passives for the current component.
- ERC rules for the current component.

- Current module virtual ports.
- Visible existing nets in the scoped view.
- Global nets visible to all modules.
- Summary of already completed components.
- Refdes counters for new passives.
- Optional verifier feedback from the previous attempt.

Wiring rules.

- Start from the provided reference circuit and adapt template role names to real numbered refdes.
- Reuse an existing visible net when its semantic role matches the intended connection.
- Create a new net only when no visible net has the required meaning.
- Connect every PowerIn and Ground pin. Add local decoupling to ground for each supply rail unless the reference circuit explicitly gives a different topology.
- Tie enable and configuration pins to the required rail or ground.
- Keep oscillator input and output sides on separate nets and add load capacitors to ground.
- For I2C or open-drain lines, ensure that the shared bus has one pull-up set to the appropriate power rail.
- For switching regulators, instantiate the inductor, feedback divider, bootstrap capacitor, input/output capacitors, and freewheeling path when required.
- Do not reference hidden pins, hidden nets, or components outside the scoped view.

Completion criterion. Return only after all pins required by the reference circuit and ERC rules have been connected. If verifier feedback is provided, repair the reported local issue and preserve previously valid connections.

Repair prompt. When a local ERC check fails, TopoScope reuses the same scoped view and appends a short repair instruction. This keeps the repair local and avoids asking the model to reconsider the entire board.

Repair Prompt

Verifier feedback. The previous wiring attempt failed local ERC verification. The following rules failed: {ERC_FEEDBACK}.

Repair instruction. Modify only the local subgraph for the current component and its required passives. Keep valid existing nets and valid existing connections unchanged. Address every reported rule violation. Do not introduce entities outside the scoped view.

Baseline prompt. The baseline receives the same task description and component knowledge as TopoScope, including pin tables, electrical types, naming rules, and reference-circuit records. It does not receive topology-guided traversal, the active-net state, virtual-port resolution, graph-operation tools, or ERC repair feedback. It must emit the full Connection Set in a single response.

Baseline Prompt

Task. Generate a complete PCB schematic netlist for the given design in one response.

Inputs.

- System functional description.
- Pre-selected ICs with pin tables.
- Electrical types and pin descriptions.
- Reference-circuit summaries.
- Naming rules.

Instructions.

Table 12: Per-design APR results for TopoScope across all evaluated models. Values are percentages. For each design, the best APR is bolded.

ID	DeepSeek-V3.2	GPT-OSS-120B	Kimi-K2.5	Qwen3.5-Plus	GLM-5	GPT-5.4	Gemini-3.1-Pro
1	90.0	91.4	87.4	97.1	94.3	97.1	**98.6**
2	97.1	**98.6**	**98.6**	97.1	95.7	**98.6**	97.1
3	96.4	**100.0**	96.4	**100.0**	**100.0**	98.2	**100.0**
4	81.7	**90.0**	73.3	88.8	**90.0**	**90.0**	**90.0**
5	87.4	95.9	**100.0**	**100.0**	97.5	**100.0**	**100.0**
6	77.6	92.1	93.6	**97.5**	90.1	97.0	96.6
7	83.7	87.4	81.4	84.3	88.6	90.7	**99.3**
8	85.7	90.9	92.6	**95.7**	95.0	**95.7**	**95.7**
9	76.4	85.7	90.6	92.9	92.9	92.9	**94.4**
10	86.1	88.9	89.2	**94.5**	92.5	91.3	90.7
11	79.1	84.5	85.9	89.5	**91.9**	90.0	88.8
12	86.6	86.6	90.8	84.8	**91.1**	85.1	88.4
13	75.6	84.1	85.7	90.5	**93.2**	89.1	90.5
14	83.4	89.4	93.1	**94.4**	**94.4**	93.7	**94.4**
15	89.1	88.8	84.8	92.3	94.4	**97.9**	95.7
16	77.7	85.0	82.7	87.8	89.8	**95.9**	89.3
17	76.5	77.1	75.3	80.9	90.7	85.9	**93.3**
18	92.2	91.6	89.8	94.3	**95.9**	91.9	94.7
19	89.8	97.8	98.0	97.0	99.0	99.0	**99.5**
20	71.4	70.8	87.0	88.8	84.7	92.3	**94.9**
Average	84.2	88.8	88.8	92.4	93.1	93.6	**94.6**

- Use only the provided active ICs. Instantiate any required passive components.
- Follow the provided reference circuits and pin definitions.
- Produce a valid bipartite netlist: pins are grouped by nets, and pins never connect directly to pins.
- Use semantic net names and strict numeric refdes.
- Include all required power, ground, bus, control, clock, feedback, bypass, and protection connections.
- Return a JSON object with components and nets. Do not include explanatory prose outside the JSON object.

A.7 Detailed Assertion-Level Results

Table 12 reports the per-design APR of TopoScope for every evaluated model, complementing the thresholded Pass@k results in Table 2. The strongest models retain high APR on hard tasks despite lower Pass@k.

A.8 Expert Validation and Threshold Calibration

We validate the assertion-based evaluation protocol against expert binary labels. We randomly sample 300 generated schematics from the benchmark outputs and invite two domain experts to label each schematic as pass or fail under the criterion of basic task satisfaction without major electrical errors. The two experts annotate independently under a double-blind protocol: they do not see model identities, generation settings, APR scores, or each other's labels. We aggregate their annotations into a final expert label for each sample, and compare the automatic labels with these final expert labels using Cohen's κ,

$$\kappa = \frac{p_o - p_e}{1 - p_e}, \tag{12}$$

where p_o is the observed agreement and p_e is the agreement expected by chance. We sweep the APR threshold from 0.800 to 1.000 and choose the threshold that maximizes κ within each tier.

For the confusion matrices, we also report precision, recall, and F1 score:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \quad \text{F1} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{13}$$

Table 13: Detailed expert validation results. Panels (a)–(d) show confusion matrices by difficulty tier and overall. Panel (e) summarizes the threshold search and derived metrics. P and F denote pass and fail labels.

(a) Easy tasks (threshold $= 0.800$, $N = 37$, $\kappa = 0.890$)

	Expert: P	**Expert: F**	**Total**
Auto: P	20 (TP)	2 (FP)	22
Auto: F	0 (FN)	15 (TN)	15
Total	20	17	37

(b) Medium tasks (threshold $= 0.890$, $N = 167$, $\kappa = 0.820$)

	Expert: P	**Expert: F**	**Total**
Auto: P	46 (TP)	9 (FP)	55
Auto: F	4 (FN)	108 (TN)	112
Total	50	117	167

(c) Hard tasks (threshold $= 0.900$, $N = 96$, $\kappa = 0.818$)

	Expert: P	**Expert: F**	**Total**
Auto: P	23 (TP)	5 (FP)	28
Auto: F	2 (FN)	66 (TN)	68
Total	25	71	96

(d) Overall (tier-specific thresholds, $N = 300$, $\kappa = 0.835$)

	Expert: P	**Expert: F**	**Total**
Auto: P	89 (TP)	16 (FP)	105
Auto: F	6 (FN)	189 (TN)	195
Total	95	205	300

(e) Summary of threshold calibration metrics

Difficulty	**Threshold**	N	κ	**Precision**	**Recall**	**F1**
Easy	0.800	37	0.890	90.9%	100.0%	95.2%
Medium	0.890	167	0.820	83.6%	92.0%	87.6%
Hard	0.900	96	0.818	82.1%	92.0%	86.8%
Overall	tier-specific	300	0.835	84.8%	93.7%	89.0%

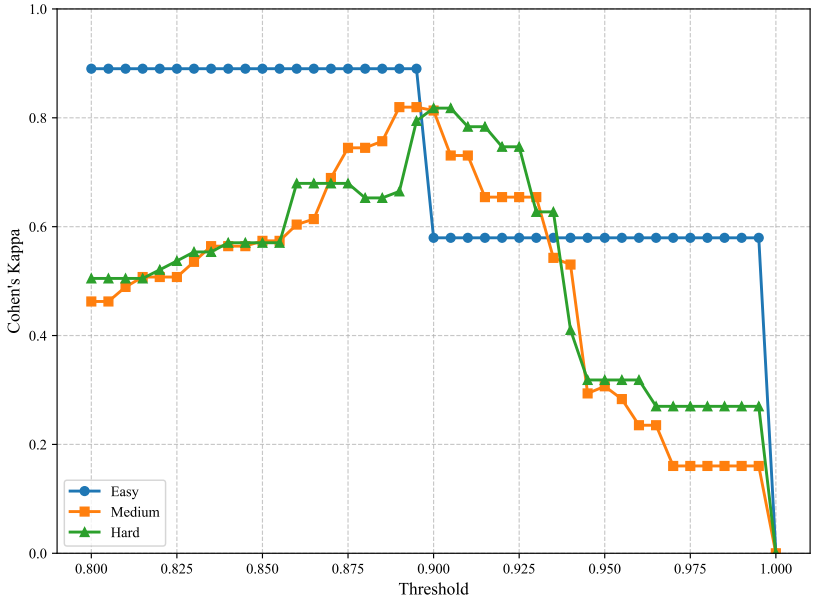


Figure 4: Cohen's κ as a function of the APR threshold for each difficulty tier. The selected thresholds are 0.800 for easy, 0.890 for medium, and 0.900 for hard.

Table 13 reports the selected thresholds and the resulting confusion matrices. The best thresholds are 0.800 for easy designs, 0.890 for medium designs, and 0.900 for hard designs. Under these tier-specific thresholds, the automatic assertion-based evaluator achieves an overall κ of 0.835 on 300 expert-labeled samples, with precision 84.8%, recall 93.7%, and F1 score 89.0%.

Figure 4 plots Cohen's κ against the APR threshold for each difficulty tier. The easy-tier curve has a broad high-agreement region: easy designs contain fewer assertions, so one missed assertion changes APR by a larger discrete amount. Medium and hard designs contain more assertions, including supporting-topology checks, producing finer APR increments and a more threshold-sensitive pass/fail boundary.

Table 14: Qualitative comparison with closely related PCB schematic generation and optimization systems.

Dimension	TopoScope	PCBSchemaGen	CircuitLM	GNN PCB schematic
Core task	End-to-end PCB schematic generation from a task specification	End-to-end PCB schematic and code generation	Natural-language to CircuitJSON schematic generation	Bypass-circuit generation around a specified IC
Primary representation	Runtime schematic graph and Connection Set	SKiDL-centered code representation with KG-backed checking	CircuitJSON with SVG visualization	Bipartite graph over schematic symbols and nets
Benchmark scale	20 full designs; hard tier spans about 120–200 pins	23 published tasks; hard tasks exceed 100 component pins	100 embedded-system prompts, mainly prototype-scale	Released examples, without a shared end-to-end test set
Generation control	Module planning followed by component-scoped wiring	Prompted code generation with verification feedback	Sequential multi-agent pipeline	Supervised GNN inference on an existing graph
Context management	Graph-derived scoped view exposes only the current pins, active nets, global nets, and virtual ports	Component knowledge and rules are injected into the code-generation context	Retrieved component records and reasoning traces are passed between agents	Local message passing over the provided graph
Verification and evaluation	In-loop ERC patterns plus post-generation functional assertions, APR, Pass@k, HR, and expert-calibrated thresholds	Syntax/ERC/SI-based verification and expert validation	Hybrid library-compliance and electrical-logic scoring	AUPRC or regression error for insertion prediction
Main distinction	The schematic graph is an operative runtime structure for scoping, verification, and evaluation	Datasheet-derived KG and SI verification ground executable schematic code	Multi-agent visual schematic generation for prototyping	Learning-based optimization support rather than generative design

A.9 Comparison with Prior Work

Table 14 summarizes the task definitions and representations of TopoScope and related PCB schematic systems. The PCBSchemaGen comparison below reports aggregate results by difficulty tier.

Task and evaluation alignment. HWE-Bench's [3] board-level dataset and code were unavailable when our experiments were conducted. CircuitLM [26] starts from open-ended requests and retrieves components, often specified as boards or generic modules; TopoScope takes exact, pre-selected ICs with datasheet-derived pin tables, reference circuits, and ERC rules. Applying TopoScope to CircuitLM's 100 prompts requires component selection, record construction, and conversion from its module/pin ontology and CircuitJSON pairwise connections to our pin–net graph. CircuitLM combines generic deterministic ERC with an LLM judge for prompt compliance, whereas TopoScope uses task-specific functional assertions. The GNN-based method [10] targets bypass-circuit generation around an IC; its released artifact provides generated examples rather than an end-to-end schematic test set.

Comparison with PCBSchemaGen. PCBSchemaGen [27] supplies fixed components and deterministic task-level checks, which we translate into our component records and assertion format. We evaluate TopoScope on 12 tasks selected from the same larger task pool, using different samples from the 23 tasks in its published Table 2. The eight medium tasks are `PCBSG-07` (isolated bipolar supply), `15` (isolated gate driver), `34` (STM32 minimum system with CAN), `35` (STM32 minimum system with crystal/UART), `38` (RS485 transceiver), `42` (SPI flash), `43` (I2C EEPROM), and `45` (buck/LDO power tree). The four hard tasks are `PCBSG-49` (5S BMS with CAN), `52` (IoT sensor node with CAN), `54` (Li-ion charger and power monitor), and `56` (SPI Ethernet interface). We compare the medium aggregate against its Tasks 7–16 and the hard aggregate against its Tasks 17–23, excluding easy tasks.

Table 15 reports unweighted task averages of Pass@1 and Pass@5. All DeepSeek-V3.2 tasks and all eight Qwen3-Max medium tasks have 15 real runs. Qwen3-Max hard tasks have 11 real runs for `PCBSG-49` and seven each for `52`, `54`, and `56`; the remaining 28 slots are filled by sampling with replacement from each task's original seven runs (seed 2026), not by additional independent trials. Pass@k is computed as $1 - \binom{15-c}{k} / \binom{15}{k}$. The 95% intervals use a task-and-run percentile bootstrap with 20,000 replicates (seed 2026), separately within each difficulty tier. Published success counts are reconstructed from the reported Pass@1 values and $n = 15$, and their intervals are computed using the same bootstrap. The Qwen3-Max hard intervals are conditional on the completed sample.

Table 15: Difficulty-matched comparison with PCBSchemaGen Table 2. Point estimates with upper/lower 95% CI errors (%). Task samples differ across sources, and TopoScope is scored with our translated deterministic assertion protocol. PCBSchemaGen intervals are reconstructed from aggregate success counts.

Tier	Model	PCBSchemaGen Pass@1	TopoScope Pass@1	PCBSchemaGen Pass@5	TopoScope Pass@5
Medium	Qwen3-Max	$63.3^{+23.4}_{-25.3}$	$80.8^{+14.2}_{-16.6}$	$79.0^{+20.5}_{-29.2}$	$99.5^{+0.5}_{-5.0}$
	DeepSeek-V3.2	$76.0^{+18.0}_{-22.0}$	$76.7^{+13.3}_{-16.7}$	$89.2^{+10.8}_{-20.9}$	$98.9^{+1.1}_{-7.3}$
Hard	Qwen3-Max	$13.3^{+15.3}_{-12.3}$	$81.7^{+18.3}_{-28.4}$	$36.6^{+30.5}_{-31.8}$	$99.0^{+1.0}_{-9.7}$
	DeepSeek-V3.2	$57.1^{+25.8}_{-28.6}$	$73.3^{+15.0}_{-16.6}$	$76.2^{+23.8}_{-33.4}$	$99.8^{+0.2}_{-3.7}$

On medium tasks, TopoScope's Pass@1 is 80.8% for Qwen3-Max and 76.7% for DeepSeek-V3.2, compared with 63.3% and 76.0% in the published results. On hard tasks, the corresponding values are 81.7% and 73.3%, compared with 13.3% and 57.1%. These tier-level scores summarize different task subsets and evaluation protocols. Mechanistically, PCBSchemaGen primarily verifies and repairs generated code, whereas TopoScope additionally uses TCS to scope each component-level wiring decision before generation.

A.10 Hallucination Rate Definition

Hallucination rate (HR) is a run-level diagnostic metric. It measures whether the generated schematic contains any invalid structural entity or physically impossible pin state before APR is interpreted. For a set of generated runs $\mathcal{R}$, we define

$$\mathrm{HR} = \frac{1}{|\mathcal{R}|} \sum_{G \in \mathcal{R}} \mathbb{1}\left[H_{\mathrm{pin}}(G) \vee H_{\mathrm{multi}}(G) \vee H_{\mathrm{float}}(G)\right]. \tag{14}$$

The three indicators correspond to the failure modes below. APR measures functional assertion satisfaction, while HR measures violations of graph and pin-level validity.

Invalid pin reference. This category covers both invented pins and incorrect pin references. A pin reference is invalid if the referenced component is not present in the design, if the pin name is absent from the component's datasheet-derived pin table, or if the generated reference uses a malformed pin identifier that cannot be resolved to a physical pin. For example, connecting to a nonexistent MCU pin or using a package-incompatible pin name is counted as an invalid pin reference. TCS reduces this error mode by only exposing pins from the focused component and visible graph state.

Multi-net pin. A generated schematic violates the single-net property if the same physical pin is assigned to two or more distinct nets. This state cannot be realized by a valid netlist, and it often corresponds to an unintended short or an inconsistent repair. The detector builds a map from pin identifiers to their assigned nets and flags any pin whose net set has cardinality greater than one. This check is performed before functional assertions, because assertion matching assumes that the generated Connection Set defines a valid pin-to-net relation.

Required-but-floating pin. This category covers pins that must be connected but have no assigned net. It includes power pins, ground pins, enable or configuration pins that are required by the component record, and terminals of instantiated passive components. Passive terminals are included because a resistor, capacitor, inductor, diode, or crystal with a floating terminal creates a dangling subgraph rather than a meaningful circuit element. The detector uses the component knowledge schema and the generated component list to identify required pins, then checks whether each required pin appears in at least one net.

A.11 Limitations

Broader Impacts. TopoScope is intended as a design-assistance and evaluation framework for schematic-level PCB generation. Its positive impact is to lower the barrier for early-stage hardware prototyping, make LLM-generated schematics more reproducible to evaluate, and provide a deterministic assertion-based protocol for studying functional correctness in real-IC PCB designs. By

exposing pin-level and topology-level errors earlier in the generation process, the framework may also reduce avoidable design iterations and support safer use of LLMs in engineering workflows.

At the same time, generated schematics should not be treated as production-ready hardware without expert review. Incorrect schematic generation can lead to non-functional boards, component damage, or safety risks in high-current, high-voltage, medical, automotive, or other regulated settings. The release of the benchmark, assertions, and evaluation scripts is therefore intended to support research and design review, not to replace professional engineering sign-off, downstream EDA checks, simulation, layout verification, and hardware validation.

Structured component-data construction. TopoScope relies on structured component records that include pin tables, reference circuits, and ERC rules. Building these records from datasheet PDFs requires careful extraction and normalization. Foundation models can substantially reduce this effort, but the resulting schemas still benefit from consistency checks, especially when datasheets are ambiguous or vendor-specific terminology is used. In this paper, the component records used in the experiments were processed with a frontier general-purpose LLM and checked before evaluation.

Rule evolution and verifier maintenance. The ERC framework is designed to be extensible: adding a new rule type usually requires a matcher and the corresponding rule entries in component records. When a new rule becomes useful for existing components, those records may need to be updated so that the rule can be applied consistently. The matcher itself also needs careful design, since PCB schematics often admit valid implementation variants. In practice, this calls for rule templates that are precise enough to catch real violations while retaining tolerance for equivalent topologies.

Output portability and packaging details. TopoScope outputs a component netlist and schematic graph that can be deterministically converted into common netlist representations, including code-based formats such as SKiDL. This makes the electrical connectivity portable across downstream tools. However, the current work does not address footprint or silkscreen selection. In practical PCB workflows, selecting an appropriate package, footprint, and silkscreen depends on application-specific constraints such as assembly process, component availability, board density, thermal requirements, and manufacturer design rules. These packaging and layout-facing decisions are outside the scope of this paper, which focuses on schematic-level connectivity under real-IC pin and topology constraints.

Benchmark scope. The benchmark covers 20 designs across three tiers and six domains, with increasing pin count and multi-IC connectivity. It is designed to stress heterogeneous real-IC schematic topology rather than to exhaust the full diversity of industrial PCB design. It does not cover footprint selection, placement, routing, BOM optimization, manufacturability, or board-level layout closure. Future benchmarks can extend this foundation with larger component libraries, richer analog and power-electronics parameter constraints, and closer integration with downstream layout and simulation tools.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction state the main contributions and scope, including TopoScope, TCS, pattern-based verification, and the 20-design benchmark. These claims are supported by the method and experiments in Sections 3 and 4.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The paper includes a concise limitations paragraph in the main text and a detailed discussion in Appendix A.11. The discussion covers component-data construction, ERC maintenance, evaluation portability, topology-level verification, and benchmark scope.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [N/A]

   Justification: The paper does not present theoretical results in the form of theorems or formal proofs. The equations in the paper define graph representations, pattern matching, APR, and evaluation metrics.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The paper describes the benchmark, model set, prompt templates, graph representation, evaluation metrics, repeated-run protocol, and expert-calibrated thresholds. Anonymized code and data are also provided to support reproduction of the main results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide anonymized code and data with the submission, including benchmark specifications, component schemas, functional assertions, prompt templates, and evaluation scripts needed to reproduce the main results.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: The paper does not train new models. It specifies the evaluated hosted LLMs, decoding setup, number of runs, baseline and ablation conditions, Pass@k estimator, APR computation, and threshold calibration in Section 4.1 and the appendix.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

Justification: The experiments use repeated samples with $n = 7$ per model and task, and Pass@k is computed with the unbiased estimator. The paper also reports APR, per-design breakdowns, Cohen's κ, and threshold-sweep validation against expert labels.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: The experiments use hosted LLM APIs from the evaluated model providers and local CPU-based graph validation and assertion evaluation. The paper reports the model providers, experimental protocol, number of runs, and evaluation procedure in Section 4.1 and the appendix.

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The research uses synthetic benchmark tasks, generated schematic artifacts, and technical expert review, and does not involve personal data, sensitive attributes, or high-risk human-subject settings. We have reviewed the NeurIPS Code of Ethics and found no deviation from it.

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The paper discusses positive impacts for engineering productivity and reproducible schematic evaluation, as well as risks from over-reliance on generated schematics without expert review. These issues are discussed in the limitations paragraph and Appendix A.11.

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: The paper does not release high-risk pretrained models, image generators, scraped internet datasets, or other assets that require special release safeguards. The released materials are benchmark specifications, schemas, assertions, prompts, and evaluation code.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: The paper uses existing LLMs accessed through official providers or documented release channels and cites the corresponding model reports or model cards. We do not redistribute these models, proprietary model assets, or third-party datasets.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: The paper introduces new benchmark assets, component schemas, ERC rule definitions, functional assertions, prompt templates, and evaluation scripts. These assets are documented in Appendix A.1–A.6 and are included in the anonymized submission package.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [N/A]

    Justification: The work does not involve crowdsourcing or human-subject experiments. The expert validation uses independent technical review of generated schematic artifacts and does not collect personal data or study human behavior.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The work does not involve crowdsourcing or human-subject experiments. The expert validation concerns schematic artifacts rather than human participants, so IRB approval is not applicable.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are a core component of the proposed agent and the empirical evaluation. The paper describes their role in generation, lists the evaluated models, and provides the prompt templates in Appendix A.6.