PCBRouteAgent: How Close Are LLM Agents to Becoming PCB Routing Engineers?

Yue Zhang[*1], **Jilei Sun**[*1], **Liqiang Jing**[*,†1], **Yijie Wei**[‡2] **and Ruohan Yang**[3]

[1]The University of Texas at Dallas, [2]Texas Instruments Inc., [3]Iowa State University

[*] Equal contribution. [†] Corresponding author: Liqiang Jing (jingliqiang6@gmail.com).
[‡] This work was conducted outside the author's role at Texas Instruments Inc.

Abstract: Automating printed circuit board (PCB) routing requires coordinating board-level connectivity decisions with geometric execution under strict design rules. Although recent agents can interact with native routing tools, reliable routing on complex boards remains challenging. We present **PCBRouteAgent**, a multimodal PCB routing agent inspired by how human engineers coordinate global strategy with local execution. A Global Routing Planner combines board images and structured electrical and geometric information to specify connection topology, routing order, corridors, and layer transitions. A Detailed Routing Executor realizes these guides through KiCad's native routing engine and performs bounded local recovery. Unresolved failures are returned as structured feedback, enabling targeted guide revisions while preserving accepted routes. We evaluate PCBRouteAgent on 247 synthetic and real-world boards from four PCBWorld-Bench test splits. It achieves clean-pass rates of 100%, 94.9%, 100%, and 40% on D2, D3-A, D3-B, and D3-C, respectively. On the three real-world subsets, PCBRouteAgent outperforms the best evaluated baseline in clean-pass rate by 13.9, 36, and 10 percentage points, respectively. Case studies reveal limitations in routing-order planning and signal-specific constraints, highlighting a substantial remaining gap between connection completion and electrically validated PCB designs. The project page and code are available at https://chip-ai-agent.github.io/research/pcbrouteagent and https://github.com/chip-ai-agent/PCBRouteAgent, respectively.

1. Introduction

Printed circuit boards (PCBs) provide mechanical support and electrical interconnections for electronic components. Before a circuit design can be fabricated as a PCB, its schematic connectivity must be translated into a physical board layout. This process includes component placement, which determines the positions and orientations of component footprints, followed by routing, which realizes the required electrical connections. As illustrated in Figure **1**, routing connects the pads belonging to each electrical net* through copper traces and, when necessary, vias that enable transitions between copper layers. A net may contain two or more pads, all of which must form a connected network while remaining electrically isolated from other nets. The resulting routing must satisfy design rules governing clearance, trace width, vias, and board boundaries, as well as signal-specific constraints such as differential-pair requirements [1, 2]. Beyond establishing valid connections, routing also seeks to limit wirelength and unnecessary layer transitions while preserving space for other nets.

*A net is an electrical connectivity group in a PCB design, specifying the component terminals that must be electrically connected according to the circuit schematic.

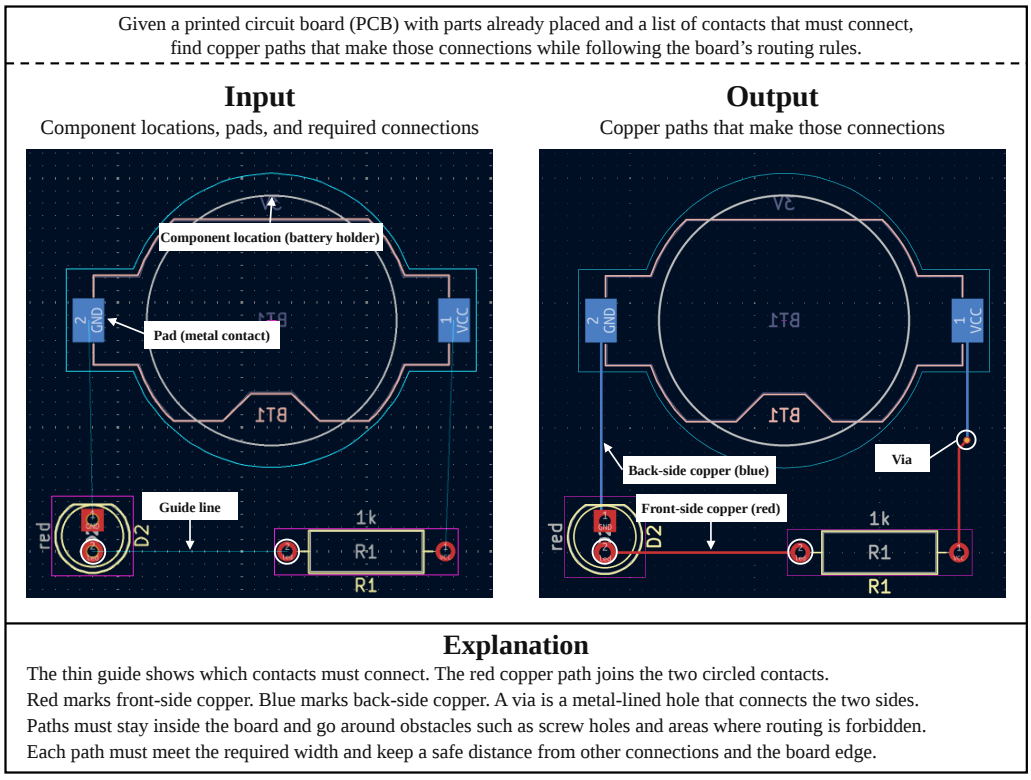


Figure 1: PCB routing connects required pads with copper traces and vias while satisfying design rules.

These requirements make PCB routing a coupled physical design problem in which each routing decision affects the connections that remain to be completed.

Routing complex PCBs remains labor-intensive, motivating automatic routers that reduce the need for manual trace construction. Classical rule-based methods use heuristic search [3], net ordering, and congestion-aware costs [1] to construct routes under geometric constraints. Although these methods automate much of the process, complex boards can still require careful tuning and manual intervention [2]. To reduce this reliance on handcrafted decisions, learning-based approaches have been explored for routing tasks, including global routing [4], net ordering [5], and PCB fanout [6]. Their effectiveness, however, depends on how routing decisions are represented and connected to physical execution. As discussed in PCBWorld [2], grid-based formulations face rapidly growing decision spaces as routing resolution increases, while approaches that guide an existing autorouter leave much of the physical routing process to that router. These limitations motivate studying agents that interact directly with a design engine and adapt their decisions using execution feedback. PCBWorld [2] provides such an environment by exposing KiCad's native routing operations and design-rule feedback to reinforcement-learning policies and tool-using LLM agents. This interaction allows agents to observe the consequences of routing decisions and revise subsequent actions as the board evolves.

Despite this progress, reliable agent-based PCB routing remains challenging. The LLM agents evaluated in PCBWorld [2] still lag behind rule-based routers in completing connections while satisfying design rules, particularly on complex real-world boards. Providing access to native routing opera-

tions and execution feedback does not, by itself, ensure effective board-level coordination. An agent must decide which connections to prioritize, allocate limited routing space across competing nets, and determine when a layer transition is necessary. These decisions are coupled: a locally feasible route may block subsequent connections, while a connection that repeatedly fails may require a change in its topology or layer assignment rather than another local attempt. Moreover, as routing proceeds, the agent must distinguish difficulties that can be resolved locally from those requiring a broader strategy revision. These challenges highlight the need for an explicit mechanism that coordinates board-level planning with geometric execution and uses routing failures to revise the plan.

To address these limitations, we propose **PCBRouteAgent**, a multimodal PCB routing agent inspired by how human routing engineers coordinate global strategy with local execution. Engineers consider connection priorities, routing regions, and layer usage while constructing individual traces, and revisit these decisions when local obstacles prevent further progress. Following this principle, PCBRouteAgent integrates a **Global Routing Planner** and a **Detailed Routing Executor** through a feedback-driven planning–execution loop. The planner combines board images with structured electrical and geometric information to organize connectivity demands into routing guides specifying connection topology, routing order, corridors, and layer transitions. The executor translates these guides into physical traces and vias through KiCad's native routing engine, using bounded local recovery to handle geometric difficulties. When recovery is insufficient, it returns structured execution evidence to the planner, which generates targeted guide revisions while preserving accepted routes. This design enables board-level guidance to adapt to the physical constraints revealed during execution.

We evaluate PCBRouteAgent on PCBWorld-Bench [2], comprising 128 synthetic boards and 119 real-world boards across four test splits. PCBRouteAgent achieves a 100% clean-pass rate on D2. On the real-world D3-A, D3-B, and D3-C subsets, PCBRouteAgent achieves clean-pass rates of 94.9%, 100%, and 40%, respectively, exceeding the best evaluated baseline on each split by 13.9, 36, and 10 percentage points. These baselines achieve 81%, 64%, and 30% clean pass: PCBWorld PPO w/o finish [2] on D3-A, and Freerouting [7] on D3-B and D3-C. It also outperforms the published PCBWorld GPT-5.4 agent [2, 8], whose clean-pass rates are 42% on D3-A and 0% on D3-B. On D3-C, PCBRouteAgent reaches 94.9% mean routability and reduces the average design-rule violation count from Freerouting's 32.0 to 6.3 per board. Despite these gains, our case studies reveal important limitations (Appendix D). Earlier routing decisions can block subsequent connections, while fully connected outputs can overlook differential-pair relationships. A substantial gap therefore remains between strong benchmark performance and the broader requirements of PCB engineering.

2. Related Work

2.1. PCB Routing

PCB routing connects the pads specified by a netlist while respecting the geometry of components, existing copper, board boundaries, and electrical design rules. Classical approaches search for feasible paths and then refine them according to costs such as wirelength, congestion, and the number of vias. In IC routing, TritonRoute [9] illustrates the global–detailed decomposition: global routing determines coarse topology and layer guidance, while detailed routing realizes traces and vias un-

der geometric design rules. Lee's maze-routing algorithm [3] established a systematic way to find connections in a discretized search space; subsequent PCB routers have incorporated constraints that simple shortest-path search cannot capture. For example, Lin et al. [1] address complicated board constraints and differential pairs within a unified routing algorithm. These methods demonstrate the value of explicit geometric search and rule-aware optimization, but they also make clear that routing quality depends on decisions made across multiple scales: the order in which nets are attempted, the space reserved for future connections, and the local shape of each trace. Learning-based work has targeted different parts of this decision process. Liao et al. [4] investigate deep reinforcement learning for global routing, while FanoutNet [6] learns decisions for the specialized PCB fanout problem. GPCB [10] uses a generative pretrained Transformer to predict routing-related network-flow patterns and then constructs physical routes from those predictions. More recently, world-model reinforcement learning [5] has been integrated with Freerouting so that learned guidance can be combined with an existing autorouter. These systems differ in what they predict and in how much of the routing procedure is handled by a separate search or routing engine. PCBWorld [2] provides another setting: it exposes KiCad's native routing operations to reinforcement-learning policies and tool-using LLM agents, and evaluates the resulting board with engine-based design-rule checks. This interactive setting makes it possible to observe how a routing decision changes the board before choosing the next action. Our work uses that setting to study the agent's ability to coordinate routing decisions over an evolving board, rather than treating each connection as an isolated path-finding problem.

2.2. LLM Agents for EDA

LLM agents have begun to use EDA tools as part of longer design workflows, where a model plans tool calls and uses their feedback to guide later steps. ReAct [11] interleaves reasoning with environment actions, while Reflexion [12] uses feedback and reflective memory to improve subsequent attempts without updating model weights. Such workflows require the agent to track design state, choose suitable tools, and recover when a tool reports an error. ChatEDA [13] connects language-model planning to EDA tool invocation in an RTL-to-GDSII flow, and EDAid [14] investigates collaboration among multiple agents for electronic design automation. VerilogCoder [15] focuses on HDL development, using planning and feedback from code-oriented tools, including waveform tracing, to revise generated Verilog. Although these systems address different stages of hardware design, they share a useful pattern: the agent proposes an action, receives evidence from a domain tool, and uses that evidence to decide what to do next. PCB layout presents a different physical interface to this pattern. PCBAgent [16] combines an RL-based placement agent with an LLM agent for interactive component-placement optimization; placement choices affect routability, but placement alone does not test whether an agent can complete the required copper connections. Recent PCB benchmarks have made such distinctions more visible. PCB-Bench [17] evaluates LLMs on PCB placement and routing tasks, including the interpretation of board information. OmniRouting [18] assesses routing reasoning under geometric, design-rule, and electrical constraints, and includes a tool-augmented task. These evaluations help characterize what a model understands or proposes, while an interactive routing environment additionally reveals whether a sequence of actions yields a valid board as the geometry changes. PCBWorld [2] supplies this engine-grounded interface and evaluates completed boards with native design-rule feedback. PCBRouteAgent therefore studies an LLM agent in the closed loop between planning, tool execution, board-state observation, and validation. The emphasis is on the agent's ability to maintain global routing objectives while adapting its next decision

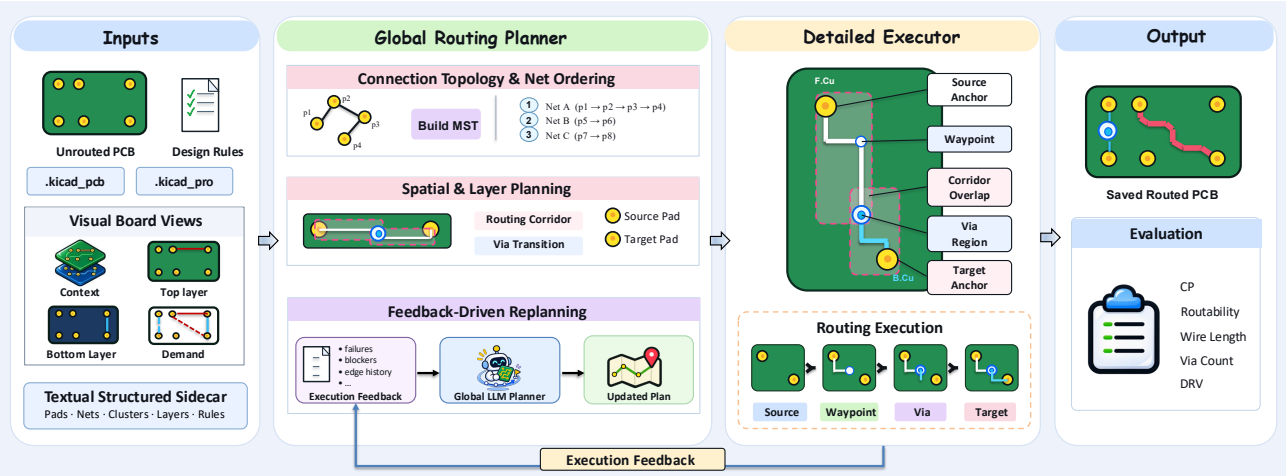


Figure 2: Overview of PCBRouteAgent.

to the board produced by earlier actions.

3. Methods

From the perspective of a PCB routing engineer, routing requires decisions at two spatial scales: coordinating connection priorities, routing regions, and layer usage across the board, and realizing individual traces around pads and obstacles. A locally feasible connection may consume space needed by another net, while a promising board-level strategy may prove geometrically infeasible during execution. This motivates separating global planning from detailed routing and revising the plan when local difficulties arise. A similar separation is used in IC routing flows such as OpenROAD [19], where global routing produces guides for detailed routing.

We adopt this principle for PCB routing. As illustrated in Figure **2**, PCBRouteAgent is a multimodal agentic framework for PCB routing that integrates two tightly coupled components: a Global Routing Planner and a Detailed Routing Executor. The Global Routing Planner takes on the board-level planning responsibilities of a human PCB routing engineer, developing and maintaining a high-level routing strategy that covers key decisions such as connectivity planning, routing priorities, and spatial and layer planning. Guided by this strategy, the Detailed Routing Executor establishes physical connections through KiCad's native push-and-shove (PNS) router [20, 2] while handling local obstacles and geometric constraints, generating structured execution feedback when routing attempts fail. The Global Routing Planner then leverages this feedback and multimodal board context to revise its strategy and guide subsequent execution, enabling the agent to iteratively adapt its routing decisions. The overall procedure is summarized in Algorithm 1 in Appendix A.

3.1. Task Formulation

Given an initial PCB B_0, a target net set $\mathcal{N}$, and design rules $\mathcal{R}$, the routing task is to produce a routed board $B_r = \mathcal{F}(B_0, \mathcal{N}, \mathcal{R})$, where $\mathcal{F}$ denotes the routing agent that maps the input board,

target nets, and design rules to the resulting board. Here, B_0 specifies fixed component placements, pad geometries, copper layers, obstacles, and existing routing; $\mathcal{N}$ specifies the required electrical connections; and $\mathcal{R}$ defines track widths, clearances, and via constraints. The agent accesses these inputs through board images and structured board information. The output B_r contains physical traces and vias saved in a KiCad PCB file. The objective is to connect all pads belonging to each target net while satisfying the design rules and keeping wirelength and via count low.

3.2. Global Routing Planner

Routing multiple nets on a shared PCB requires coordinating limited routing space while maintaining electrical connectivity. To address this challenge, the Global Routing Planner first organizes disconnected components within each net into a connection topology and then assigns spatial routing guides to support their physical realization. The resulting routing guide provides the Detailed Routing Executor with a structured plan for establishing these connections.

3.2.1. PCB BoardRepresentation

Effective global routing requires understanding both the outstanding connectivity demands and the physical constraints under which these connections must be established. Given the initial PCB B_0, target net set $\mathcal{N}$, and design rules $\mathcal{R}$, we construct a multimodal board representation comprising **board images** $\mathcal{I}$ and **structured text** S. Through PCBWorld's KiCad-based engine [2, 20], we extract pad geometries, copper layers, obstacles, and existing routing from B_0, target-net connectivity from $\mathcal{N}$, and routing constraints from $\mathcal{R}$. This information is represented in two complementary forms. The **visual modality** $\mathcal{I} = \{I_{\text{layout}}, I_{\text{front}}, I_{\text{back}}, I_{\text{demand}}\}$ depicts the overall board layout, front- and back-copper states, and outstanding connectivity demands. The **textual modality** S serializes pad coordinates and geometries, layer information, net assignments, connectivity clusters, and design rules. For each target net $n \in \mathcal{N}$, electrically connected pads form clusters $\mathcal{C}_n = \{C_{n,1}, \ldots, C_{n,K_n}\}$, whose remaining connections define the routing demands. Together, the images provide spatial context, while the structured text supplies precise identifiers and numerical constraints for global planning.

3.2.2. Connection Topology and Routing Order

Although the board representation identifies the remaining connectivity demands, it does not specify how disconnected clusters should be connected. For a net containing multiple clusters, numerous connection topologies can satisfy the same electrical requirements, while independently connecting every pair would introduce unnecessary routing. To avoid exhaustively searching this combinatorial space, we formulate connectivity planning as a minimum spanning tree (MST) [21] problem over the disconnected clusters, yielding a compact initial topology. For each target net n, we construct a weighted graph $G_n = (\mathcal{C}_n, \mathcal{E}_n, w_n)$, where vertices represent connectivity clusters and edges represent candidate inter-cluster connections. Each candidate edge $e \in \mathcal{E}_n$ is assigned a weight $w_n(e)$ based on the minimum Euclidean distance between pads in the corresponding clusters, with an additional penalty for incompatible pad layers. The resulting MST $T_n = (\mathcal{C}_n, \mathcal{E}_n^*)$ specifies $K_n - 1$ connection

tasks spanning the K_n clusters. It minimizes the estimated total cost under this heuristic, while leaving the physical realization of these connections to subsequent routing stages.

Beyond determining the connection topology within each net, the planner must also coordinate routing across multiple nets that share limited board space. We therefore initialize the routing order using the total estimated connection length of each net's spanning tree, prioritizing shorter nets first. This provides a lightweight starting strategy for organizing routing tasks, without requiring an exhaustive search over possible execution orders. Both the connection topology and routing priorities remain adjustable as subsequent execution reveals routing constraints not captured by the initial estimates.

3.2.3. Spatial and Layer Planning

While the connection topology specifies which clusters should be connected, it does not determine how these connections should traverse the physical board. Directly prescribing exact trace geometries at the global planning stage would require resolving fine-grained obstacles and design-rule constraints before routing execution. Instead, we separate high-level spatial guidance from low-level geometric realization, allowing the Global Routing Planner to guide routing decisions while preserving flexibility for local obstacle avoidance. Specifically, for each connection edge e, we use the endpoint pads selected during topology construction to initialize ordered corridor guidance $\mathcal{Q}_e = (\mathcal{Q}_{e,1}, \ldots, \mathcal{Q}_{e,m_e})$, where m_e is the number of corridor stages for connection e. When the endpoints share a copper layer, we construct a single corridor $\mathcal{Q}_{e,1}$ from their axis-aligned bounding box, expanded by a margin δ in each direction. The resulting rectangle provides spatial guidance while leaving the exact trace geometry to the Detailed Routing Executor.

Beyond spatial guidance, routing may also require transitions between copper layers when the endpoints cannot be connected on a common layer. We split the connection at its midpoint and construct two layer-specific corridors, $\mathcal{Q}_{e,1}$ from the source pad to the midpoint and $\mathcal{Q}_{e,2}$ from the midpoint to the target pad, using the same bounding-box expansion. A via-transition region $\mathcal{V}_e$ is initialized by expanding the midpoint by δ in each direction. Rather than fixing an exact via location, the planner specifies a candidate region in which the Detailed Routing Executor can search for a feasible transition using KiCad's native routing engine. The resulting connection topology, routing order, and spatial and layer guidance are consolidated into an initial routing guide, which is validated and translated into routing tasks for the Detailed Routing Executor.

3.2.4. Feedback-Driven Replanning

Although heuristic initialization provides an efficient starting point, its simplified cost estimates and spatial guidance cannot fully anticipate the geometric constraints encountered during physical routing. Connections that appear feasible at the planning stage may fail due to obstacles, restricted routing space, or layer constraints. To address this gap, the Global Routing Planner revises its routing strategy using evidence obtained from detailed execution.

When a connection remains unresolved after the Detailed Routing Executor's local recovery attempts (Section 3.3.2), the executor produces structured feedback $\mathcal{F}_e$ for each unresolved connection e, describing the failed attempts, last reached positions, potential obstacles, and deviations from the original guide. These feedback packets are aggregated and provided to the multimodal LLM used by

the Global Routing Planner, together with the initial board images $\mathcal{I}$, structured board information S, the current routing guide, and updated connectivity summaries. The initial images remain fixed across replanning rounds; execution feedback and connectivity summaries provide observations of the evolving board. By jointly considering spatial context and execution evidence, the planner can revise high-level decisions that local routing alone cannot resolve.

At replanning round t, let $\mathcal{G}_t$ denote the current routing guide, $\mathcal{D}_t$ the batch of feedback packets $\mathcal{F}_e$ for unresolved connections, and $\mathcal{K}_t$ the updated connectivity summaries. The multimodal LLM $\mathcal{M}_\theta$ generates a guide patch, a structured set of local updates to the current routing guide, as

$$\Delta\mathcal{G}_t = \mathcal{M}_\theta\big(\mathcal{I}, S, \mathcal{G}_t, \mathcal{D}_t, \mathcal{K}_t; P_{\text{sys}}, P_{\text{replan}}\big), \tag{1}$$

where θ denotes the model parameters, $\Delta\mathcal{G}_t$ is the structured output patch, and P_{sys} and P_{replan} are the system instructions and replanning prompt template, respectively. Their templates are listed in Appendices B.1.1 and B.1.2.

The planner outputs a structured guide patch specifying targeted changes to connection endpoints, routing corridors, via-transition regions, or routing order. The patch updates the affected portions of the current guide while preserving previously accepted routes outside that scope. Before execution resumes, the patch is checked for consistency with the current snapshot and guide revision, as well as valid references to clusters, corridors, and copper layers. If validation fails, the same planner may be asked to correct its response using the previous output and validation errors; the correction prompt template is listed in Appendix B.2. The validated patch is merged into the guide, and affected connection tasks are recompiled for the Detailed Routing Executor. This planning–execution loop continues until the target connections are completed, no further tasks can be scheduled, or the execution or replanning budget is exhausted.

3.3. Detailed Routing Executor

While the Global Routing Planner determines which connections to establish and where they should be routed, realizing these decisions requires resolving precise geometric constraints that cannot be fully captured by high-level guidance. The Detailed Routing Executor translates the routing guide into physical traces and vias through KiCad's native PNS router, attempts bounded local recovery when routing fails, and returns unresolved failures to the Global Routing Planner for guide revision.

3.3.1. Guided Routing Execution

The Detailed Routing Executor takes the routing guide produced by the Global Routing Planner and compiles its planned connection edges into executable tasks. For each connection edge e, a task specifies the source and target connectivity clusters, candidate pad anchors, ordered layer-specific corridors $\mathcal{Q}_e$, and optional via-transition regions $\mathcal{V}_e$. The executor uses this guidance together with the design rules $\mathcal{R}$ to select routing targets and invoke native routing operations. These tasks are scheduled according to the routing priorities established by the Global Routing Planner, with connections within each net following their predefined task order. For each task, the executor selects a source anchor from the designated cluster that lies within the initial routing corridor and supports its assigned copper layer, prioritizing candidates closer to the next routing target. A compatible target anchor is identified from the destination cluster based on the final routing stage.

Instead of prescribing a fixed trace path, the executor progressively advances through the planned routing stages by selecting intermediate waypoints. For consecutive corridors $\mathcal{Q}_{e,j}$ and $\mathcal{Q}_{e,j+1}$ on the same layer, it identifies their overlapping region and generates candidate waypoints at the region center and surrounding positions, accounting for trace width and clearance requirements. These candidates are attempted sequentially through PNS, which constructs the physical trace between the current routing position and the selected target while handling local geometric constraints. Upon a successful attempt, the executor updates its routing position and proceeds to the next stage.

When a connection requires a layer transition, the executor instead generates candidate via locations within the designated transition region $\mathcal{V}_e$, accounting for via dimensions and clearance requirements. It then attempts to establish the transition before continuing on the next copper layer. After traversing the planned routing stages, the executor attempts to reach the target cluster and verifies whether the source and target clusters have become electrically connected. This connectivity-based verification ensures that task completion reflects an established physical connection rather than merely a successful intermediate routing operation.

3.3.2. Local Recovery and Execution Feedback

Although PNS can handle many local geometric constraints, routing attempts may still fail when obstacles block the selected path, restrict access to intermediate waypoints, or prevent a layer transition. Immediately triggering global replanning for every failure would introduce unnecessary overhead, particularly when alternative local paths remain available. We therefore employ a bounded recovery strategy that progressively explores alternative routing options before escalating unresolved failures to the Global Routing Planner.

When a routing attempt fails, the executor first explores local geometric alternatives through PNS, using walkaround to navigate obstacles and shove to adjust obstructing traces when necessary. If the current target remains unreachable, it attempts alternative waypoints within the designated routing region. For more challenging same-layer obstacles, a grid-based A* search [22] identifies potential detours, which are subsequently executed through PNS. The executor can further backtrack to earlier routing stages or switch to alternative source anchors within the same connectivity cluster. These operations expand the local search space while preserving the connection topology and layer-transition regions established by the Global Routing Planner. To avoid excessive exploration, recovery attempts are bounded by retry and execution budgets.

When local recovery is exhausted or a planned layer transition cannot be completed, the executor generates a structured feedback packet $\mathcal{F}_e$ for the current connection e, describing the attempted routing operations, last reached position, potential blocking obstacles, affected routing regions, and deviations from the original guidance. Failed tasks are rolled back to a consistent checkpoint, while previously accepted connections outside the affected scope are preserved. Rather than reporting failure as a binary outcome, the executor provides actionable evidence that helps the Global Routing Planner identify which higher-level decisions may require revision. This separation allows local geometric difficulties to be addressed through efficient recovery operations, while reserving global replanning for failures that cannot be resolved within the existing routing strategy. The resulting feedback is passed to the Global Routing Planner for the feedback-driven replanning described in Section 3.2.4.

Termination and output. The execution controller terminates the routing loop when all target connections are established, no further tasks or permitted recovery attempts remain, or the execution budget is exhausted. Connections that exhaust their replanning allowance are recorded as unresolved while other tasks may continue. A checkpoint or rollback failure also triggers early termination. On termination, the system saves the last consistent board state and reports completed and unresolved connections. Routing success is determined by the official evaluator applied to the saved board.

4. Experimental Setup

Benchmarks. We follow the agent-comparison setting of PCBWorld-Bench [2]. Each instance is a board in KiCad's [20] native `.kicad_pcb` format, with reference routes removed; component placement, netlist, and design rules remain fixed. The task is to complete the connections on that board, and the final board is evaluated by KiCad rather than by the agent's own estimate of success. We use the same three test sets as the PCBWorld main comparison: 128 synthetic, two-layer gridless boards from D2-test; 99 small open-source boards from D3-A; and 10 medium open-source boards from D3-B. We additionally include the 10 large open-source boards in PCBWorld's [2] fixed D3-C evaluation subset. PCBWorld excludes one of the 100 D3-A boards because its engine interface cannot load it. The D3-B and D3-C test subsets contain two-layer boards selected across each split's pad-count range; D3-C contains boards with more than 100 pads. D2 tests routing on boards drawn from the synthetic distribution, while all three D3 splits test transfer to real designs with their original design-rule settings. The Reference rows in Tables **2**–**4** evaluate the original routed designs retained in PCBWorld's open-source board corpus [2]. Each routing input is obtained by removing the target routing while preserving component placement, the netlist, and design rules; the Reference is the retained design before this removal, scored using the same official engine-based evaluator. D3-A and D3-B Reference values are taken from PCBWorld's published Table 22, while the D3-C Reference values are evaluated locally. These rows provide an existing-design reference for comparison with automatically produced routes. We use PCBWorld's published board identifiers and split definitions to keep the comparison board-for-board.

Protocol and metrics. We use PCBWorld's KiCad-based board evaluator [2] for the reported routing quality. Published D2, D3-A, and D3-B baseline values in Tables **1**–**3** use PCBWorld's single-rollout (@1) means from Table 22. PCBRouteAgent contributes one scored board per instance; any retries of unscored model-response failures are reported separately from this evaluation convention. Clean Pass (CP) is the primary success measure: a board passes only if all required connections are complete and KiCad reports zero error-level design-rule violations. We also report Potential Gain (Pot.), the change in the benchmark's board-potential score from the unrouted state to the final board; Routability (Rout.), the fraction of required connections completed; and design-rule violation count (DRV), wirelength (WL), and via count. The main tables report routing quality. We additionally compare runtime, model calls, token usage, and API cost for our local PCBWorld and PCBRouteAgent runs under the same evaluation conditions with Claude Haiku 5.5. Both methods use the same timing scope; detailed efficiency results are provided in Appendix C. Potential incorporates connectivity, design-rule, wirelength, and via costs. These complementary metrics distinguish a complete and rule-clean board from partial connectivity or a route that is physically expensive.

Table 1: D2 routing quality (128 synthetic boards).

Method	CP ↑	Pot. ↑	Rout. ↑	DRV ↓	WL (mm) ↓	Via ↓
Freerouting	**1.000**	14.42	**1.000**	**0.00**	409.8	2.48
OrthoRoute	0.010	0.38	0.340	8.54	323.4	11.30
KiCadRoutingTools	**1.000**	15.52	**1.000**	**0.00**	373.9	8.09
PCBWorld (Haiku 5.5)	0.633	13.47	0.897	1.12	338.8	2.44
PCBWorld (GPT-5.4)	0.880	15.24	0.980	0.27	411.8	2.00
PCBWorld (GPT-5.4-mini)	0.200	6.46	0.490	4.62	**208.6**	**0.96**
PCBWorld (GPT-5.4-nano)	0.240	6.97	0.540	4.21	233.3	1.25
PCBWorld (Qwen3.5-397B)	0.140	5.81	0.480	4.64	*229.6*	*1.02*
PCBWorld (PPO)	*0.980*	*15.84*	*0.990*	0.06	458.4	2.22
PCBWorld (GRPO)	0.950	11.43	0.980	0.19	744.3	9.84
PCBWorld (PPO terminal)	0.610	12.06	0.800	1.60	380.2	1.62
PCBWorld (PPO w/o finish)	**1.000**	15.81	**1.000**	*0.01*	497.1	1.93
PCBRouteAgent (Claude Haiku 5.5)	**1.000**	**16.27**	**1.000**	**0.00**	392.4	1.99
PCBRouteAgent (Claude Opus 5.5)	**1.000**	**16.27**	**1.000**	**0.00**	392.5	1.99

Comparisons. We place PCBRouteAgent alongside the methods in PCBWorld's [2] main result table: the reference routes and the rule-based Freerouting [7], OrthoRoute [23], and KiCadRoutingTools [24] routers. Interactive PCBWorld agents use GPT-5.4 [8], GPT-5.4-mini and GPT-5.4-nano [25], and Qwen3.5-397B [26]. RL baselines use policies trained with PPO [27] or GRPO [28].

We evaluate PCBRouteAgent with Claude Opus 5.5 [29] and Claude Haiku 5.5 [30] on all four splits. We also locally evaluate the PCBWorld interactive agent with Haiku 5.5. The prompt template for the locally evaluated PCBWorld interactive baseline is listed in Appendix B.3.

To isolate the contribution of LLM-based guide revisions, we additionally evaluate a w/o LLM variant on all four splits (Table **5**). We disable the model provider in the final implementation. MST-based guide initialization, detailed execution, and scripted local recovery remain enabled; LLM-based guide revisions are omitted. All variants are scored with the same official board evaluator.

Tables **1–3** combine the published PCBWorld Table 22 (@1) baselines with our local evaluations. The PCBWorld–Haiku rows are local reproductions. PCBWorld defines D3-C but publishes no scores for it, so Table **4** contains our local evaluations. The D3-B PCBRouteAgent result includes retries of unscored model-response failures and angle repair, so it is not a strict first-attempt @1 estimate.

Implementation details. We initialize routing corridors with an expansion margin of $\delta = 1.5$ mm. The controller uses a per-board replanning budget of 12, with at most four replans per connection and five batch-replanning rounds. Local recovery allows two stage backtracks, one same-layer A* detour, and two alternative-anchor attempts per connection, with at most one rip-up per obstructing net. For Claude Opus 5.5, D2 and D3-A use a maximum output length of 32,000 tokens with medium reasoning effort; D3-B uses a 32,000-token output limit, and D3-C uses 64,000 tokens with low effort. Reported Opus outputs undergo up to five guarded angle-repair passes, accepting changes only when connectivity, design-rule checks, and wirelength do not worsen. We score all saved boards with

Table 2: D3-A routing quality (99 real-world boards).

Method	CP ↑	Pot. ↑	Rout. ↑	DRV ↓	WL (mm) ↓	Via ↓
Reference	**1.000**	**23.16**	**1.000**	**0.00**	170.2	1.18
Freerouting	0.750	22.04	0.900	1.08	156.6	0.86
OrthoRoute	0.020	-6.16	0.530	99.70	129.8	26.97
KiCadRoutingTools	0.740	20.00	0.940	2.38	142.8	3.69
PCBWorld (Haiku 5.5)	0.343	14.77	0.721	4.06	111.5	0.96
PCBWorld (GPT-5.4)	0.420	16.15	0.810	2.71	148.1	2.36
PCBWorld (GPT-5.4-mini)	0.130	8.97	0.490	7.16	**81.9**	**0.21**
PCBWorld (GPT-5.4-nano)	0.150	8.35	0.470	7.46	*83.0*	*0.31*
PCBWorld (Qwen3.5-397B)	0.190	10.98	0.590	5.75	102.7	0.47
PCBWorld (PPO)	0.680	19.22	0.870	1.91	164.9	1.09
PCBWorld (GRPO)	0.640	15.07	0.900	1.67	337.1	14.13
PCBWorld (PPO terminal)	0.650	18.23	0.860	1.95	171.0	3.50
PCBWorld (PPO w/o finish)	0.810	19.55	0.940	0.95	262.7	3.87
PCBRouteAgent (Claude Haiku 5.5)	0.929	22.86	0.990	0.14	164.0	0.78
PCBRouteAgent (Claude Opus 5.5)	*0.949*	*23.02*	*0.994*	*0.09*	165.5	0.79

PCBWorld's official evaluation settings. The potential penalizes incomplete connections, error-level design-rule violations, wirelength, and vias; warning-level issues do not affect the score. Prompt templates are provided in Appendix B.

5. Results and Analysis

Tables **1**–**3** combine the published PCBWorld [2] Table 22 (@1) baselines with our local evaluations. PCBWorld–Haiku is locally reproduced on each split, and PCBRouteAgent contributes one scored board per instance. Table **4** reports local evaluations on the fixed D3-C subset. Bold and italic values indicate the best and second-best distinct reported values, respectively, including Reference; ties share the same formatting.

Routing quality on synthetic boards. On D2 (Table **1**), both PCBRouteAgent variants achieve a clean-pass rate of 1.00 and routability of 1.000, matching the complete routing achieved by Freerouting and KiCadRoutingTools. The Opus variant obtains a potential gain of 16.27, compared with 15.84 for PCBWorld PPO and 14.42 for Freerouting. Among these fully connected solutions, it reduces wirelength and via count relative to Freerouting by 4.2% and 19.7%, respectively. Thus, on the synthetic test set, PCBRouteAgent achieves complete, rule-clean routing with competitive physical routing costs.

Routing completion on small real-world boards. On D3-A (Table **2**), the Opus variant achieves a clean-pass rate of 0.949, compared with 0.810 for the strongest evaluated baseline, PCBWorld PPO w/o finish, an improvement of 13.9 percentage points. Its routability reaches 0.994, with an average DRV count of 0.09.

Table 3: D3-B routing quality (10 real-world boards). Results include response-failure retries and angle repair.

Method	CP ↑	Pot. ↑	Rout. ↑	DRV ↓	WL (mm) ↓	Via ↓
Reference	**1.000**	*63.96*	**1.000**	**0.00**	570.3	6.70
Freerouting	0.640	58.86	0.990	11.94	533.5	3.88
OrthoRoute	0.000	-10.25	0.440	188.20	370.0	39.80
KiCadRoutingTools	0.200	40.94	0.860	44.70	472.3	11.80
PCBWorld (Haiku 5.5)	0.000	25.32	0.442	25.40	*245.7*	1.90
PCBWorld (GPT-5.4)	0.000	25.65	0.440	24.88	**240.3**	2.06
PCBWorld (GPT-5.4-mini)	0.000	23.96	0.490	23.00	408.6	*0.94*
PCBWorld (GPT-5.4-nano)	0.000	23.62	0.470	24.00	363.9	**0.36**
PCBWorld (Qwen3.5-397B)	0.000	19.02	0.410	26.84	249.6	6.24
PCBWorld (PPO)	0.210	37.60	0.650	17.33	518.8	5.06
PCBWorld (GRPO)	0.070	21.76	0.550	22.28	1054.5	27.26
PCBWorld (PPO terminal)	0.150	33.47	0.640	17.54	550.0	10.64
PCBWorld (PPO w/o finish)	0.220	35.99	0.720	15.32	1008.6	14.57
PCBRouteAgent (Claude Haiku 5.5)	*0.800*	63.20	*0.997*	*0.20*	580.0	4.90
PCBRouteAgent (Claude Opus 5.5)	**1.000**	**64.47**	**1.000**	**0.00**	591.6	4.00

Table 4: D3-C routing quality (10 real-world boards); all method results are local evaluations.

Method	CP ↑	Pot. ↑	Rout. ↑	DRV ↓	WL (mm) ↓	Via ↓
Reference	**0.900**	**147.80**	*0.938*	*8.80*	1391.0	44.00
Freerouting	0.300	132.80	0.918	32.00	1304.0	21.40
OrthoRoute	0.000	0.60	0.063	139.90	**80.0**	17.80
KiCadRoutingTools	0.000	115.60	0.778	30.50	1086.0	57.40
PCBWorld (Haiku 5.5)	0.000	31.00	0.199	94.60	*330.9*	*4.30*
PCBWorld (PPO)	0.000	52.20	0.403	72.70	1075.0	5.00
PCBWorld (GRPO)	0.000	44.80	0.353	79.00	886.0	8.30
PCBWorld (PPO terminal)	0.000	56.20	0.412	69.50	1072.0	**1.50**
PCBRouteAgent (Claude Haiku 5.5)	0.100	135.99	0.915	9.80	1352.4	17.30
PCBRouteAgent (Claude Opus 5.5)	*0.400*	*141.80*	**0.949**	**6.30**	1410.5	21.30

Routing completion on medium real-world boards. On D3-B (Table **3**), the Opus variant completes all ten boards without error-level violations, compared with a clean-pass rate of 0.64 for Freerouting.

Large-board routing remains challenging. On D3-C (Table **4**), the Opus variant achieves a clean-pass rate of 0.40, compared with 0.30 for Freerouting and 0.00 for the evaluated RL baselines and PCBWorld–Haiku agent. It reaches 0.949 mean routability and reduces the average DRV count from Freerouting's 32.0 to 6.3, an 80.3% reduction. Nevertheless, six of the ten boards remain incomplete. The gap between high average routability and lower clean-pass rate shows that connecting most targets does not ensure successful completion of an entire board.

Table 5: Planner LLM ablation with the same detailed executor and scripted local recovery. Bold and italic indicate the best and second-best distinct values within each split; ties share formatting. Lower wirelength and via count should be interpreted alongside routing completion.

Set	Planner LLM	CP ↑	Pot. ↑	Rout. ↑	DRV ↓	WL (mm) ↓	Via ↓
D2 (128)	w/o LLM	*0.961*	*16.09*	*0.995*	*0.04*	**390.7**	**1.95**
	Claude Haiku 5.5	**1.000**	**16.27**	**1.000**	**0.00**	*392.4*	*1.99*
	Claude Opus 5.5	**1.000**	**16.27**	**1.000**	**0.00**	392.5	*1.99*
D3-A (99)	w/o LLM	0.535	20.13	0.922	1.00	**150.8**	**0.22**
	Claude Haiku 5.5	*0.929*	*22.86*	*0.990*	*0.14*	*164.0*	*0.78*
	Claude Opus 5.5	**0.949**	**23.02**	**0.994**	**0.09**	165.5	0.79
D3-B (10)	w/o LLM	0.000	52.61	0.891	5.10	**508.4**	**0.90**
	Claude Haiku 5.5	*0.800*	*63.20*	*0.997*	*0.20*	*580.0*	4.90
	Claude Opus 5.5	**1.000**	**64.47**	**1.000**	**0.00**	591.6	*4.00*
D3-C (10)	w/o LLM	0.000	123.48	0.831	18.70	**1124.9**	**5.90**
	Claude Haiku 5.5	*0.100*	*135.99*	*0.915*	*9.80*	*1352.4*	*17.30*
	Claude Opus 5.5	**0.400**	**141.80**	**0.949**	**6.30**	1410.5	21.30

Framework effectiveness with the same model. We do not conduct a full evaluation of PCBWorld's interactive agent with Claude Opus 5.5 because we verified that its per-step model invocation incurs substantially higher API costs than PCBRouteAgent. Instead, we compare the two frameworks using the same Claude Haiku 5.5 backbone (Tables **1**–**4**). PCBRouteAgent achieves clean-pass rates of 1.00, 0.929, 0.80, and 0.10 on D2, D3-A, D3-B, and D3-C, respectively, compared with 0.633, 0.343, 0.00, and 0.00 for PCBWorld–Haiku. These correspond to improvements of 36.7, 58.6, 80, and 10 percentage points. Mean routability also increases from 0.897 to 1.000 on D2, from 0.721 to 0.990 on D3-A, from 0.442 to 0.997 on D3-B, and from 0.199 to 0.915 on D3-C. The same language model therefore achieves higher connectivity and board completion within PCBRouteAgent, providing evidence for the effectiveness of the overall planning–execution framework.

Contribution of LLM-based replanning. Table **5** compares the full framework with its w/o LLM variant while retaining the same detailed executor and scripted local recovery. Without LLM-based guide revisions, clean-pass rates are 0.961, 0.535, 0.000, and 0.000 on D2, D3-A, D3-B, and D3-C, respectively. Adding Opus raises these rates to 1.000, 0.949, 1.000, and 0.400, gains of 3.9, 41.4, 100, and 40 percentage points. Haiku also improves clean-pass rates on all four splits. The distinction between partial connectivity and board completion is particularly clear on D3-B and D3-C: without the LLM, mean routability reaches 0.891 and 0.831, yet no board passes cleanly. LLM-based revisions therefore contribute to completing connections left unresolved by the scripted pipeline. These gains come with increased wirelength and via count; for example, on D3-A, Opus increases mean wirelength from 150.8 to 165.5 mm and via count from 0.22 to 0.79. This ablation establishes the benefit relative to disabling LLM-based revisions, but does not compare against a dedicated rule-based replanner or a general-purpose agent using the same execution tools.

Computational efficiency under the same conditions. The local Haiku comparison also shows that PCBRouteAgent requires fewer computational resources (Appendix C). Across all 247 boards, model calls decrease from 22,529 to 172, combined input and output tokens from 105.7M to 12.0M, and API

cost from $20.72 to $4.77. With the same timing scope, cumulative per-board runtime decreases from 36.4 to 2.6 hours. These correspond to approximately 131× fewer model calls, 8.8× lower token usage, 4.3× lower API cost, and 14× lower cumulative runtime. Together with the higher clean-pass rates on all four splits, these results show that the planning–execution framework improves both routing completion and computational efficiency.

Completion and physical routing cost. Improved completion need not produce shorter routing. On D3-B, the Opus variant uses 10.9% more wirelength than Freerouting while improving clean-pass rate by 36 percentage points. On D3-C, its wirelength is 8.2% higher, with a comparable via count of 21.3 versus 21.4. These results indicate a remaining opportunity to improve routing compactness while maintaining connectivity. Wirelength and via count must also be interpreted alongside completion: methods that leave many connections unrouted can obtain low physical costs simply by constructing less copper.

Remaining gap to PCB engineering. Our case studies reveal two limitations beyond aggregate routing scores (Appendix D). First, early routing decisions can restrict access to pads and prevent later connections from being completed despite local recovery. Second, complete connectivity does not ensure that signal-specific requirements, such as differential-pair routing [31], are addressed. In the USB example (Figure **3**), PCBRouteAgent completes all 55 connections but routes D+ and D- independently, without explicitly representing their differential-pair relationship. This result calls for electrical validation beyond connectivity and basic design-rule checks. These findings indicate that approaching the capabilities of a PCB routing engineer requires stronger coordination across routing decisions and explicit modeling of signal-specific constraints.

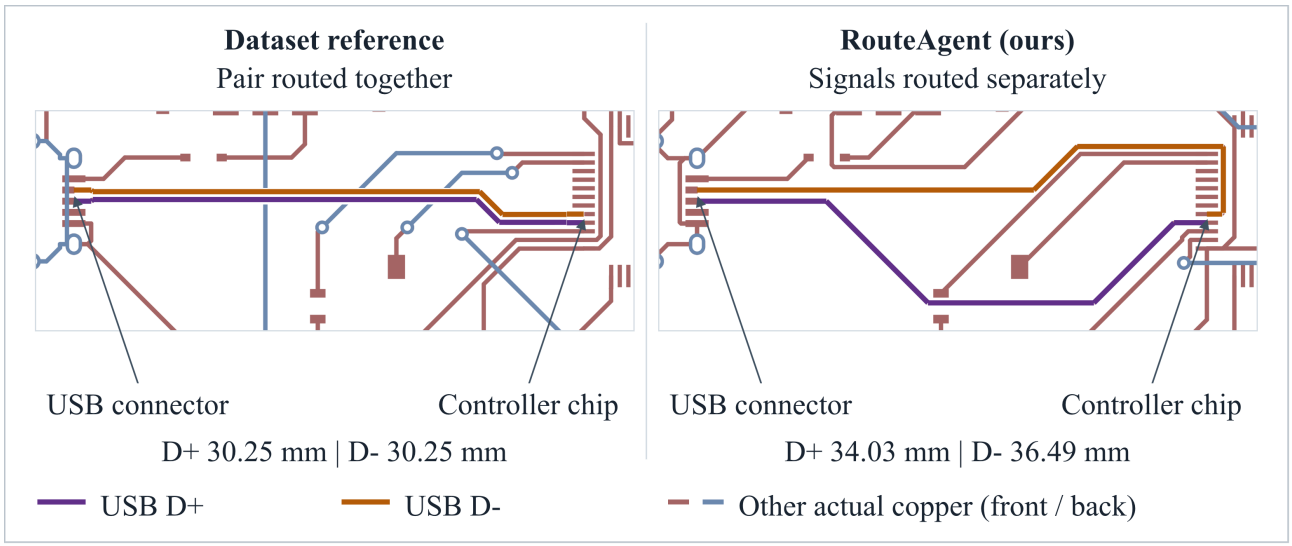


Figure 3: USB D+ and D- in the reference (left) and the fully connected PCBRouteAgent output (right). The reference routes the pair together; PCBRouteAgent produces separate paths.

6. Conclusion

We presented PCBRouteAgent, a multimodal PCB routing agent that coordinates global planning and native geometric execution through structured feedback. Inspired by the interaction between board-level strategy and local routing decisions in engineering workflows, the framework combines routing guides, bounded local recovery, and targeted guide revisions. Evaluation on 247 synthetic and real-world boards shows that PCBRouteAgent achieves complete, rule-clean routing on D2 and D3-B and improves clean-pass rate over the best evaluated baseline on all three real-world subsets. Large-board completion and routing compactness remain important challenges. Future work will investigate more effective spatial and layer planning, adaptive recovery budgets, and routing refinement that reduces physical cost while preserving completed connections.

References

[1] Ting-Chou Lin, Devon Merrill, Yen-Yi Wu, Chester Holtz, and Chung-Kuan Cheng. A unified printed circuit board routing algorithm with complicated constraints and differential pairs. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, pages 170–175, 2021.

[2] Hyungseok Song, Junseok Park, Won-Seok Choi, Seohui Bae, Han-Seul Jeong, Youngjoon Park, and Soonyoung Lee. PCBWorld: A benchmark environment for engine-grounded PCB design automation. In *KDD 2026 Workshop on Evaluation and Trustworthiness of Agentic AI*, 2026. arXiv:2607.05915.

[3] C. Y. Lee. An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers*, EC-10(3):346–365, 1961.

[4] Haiguang Liao, Wentai Zhang, Xuliang Dong, Barnabás Póczos, Kenji Shimada, and Levent Burak Kara. A deep reinforcement learning approach for global routing. *Journal of Mechanical Design*, 142(6):061701, 2020.

[5] Yin-Chi Liao, Sheng-Xin Pan, and Po-Jui Chiang. Automation of PCB autorouting via world-model reinforcement learning and freerouting integration. *Expert Systems with Applications*, 311:131424, 2026.

[6] Haiyun Li, Jixin Zhang, Ning Xu, and Mingyu Liu. FanoutNet: A neuralized PCB fanout automation method using deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 8554–8561, 2023.

[7] Freerouting Project. Freerouting: Open-source PCB autorouter. Software repository, 2026.

[8] OpenAI. Introducing GPT-5.4, 2026.

[9] Andrew B. Kahng, Lutong Wang, and Bangqi Xu. TritonRoute: The open-source detailed router. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(3):547–559, 2021.

[10] Jienan Chen, Jun Tu, Hao Wang, Jie Zheng, Hao Liu, Shenglong Bai, Xiantuo He, and Weikang Qian. GPCB routing: Generative pretrained transformers-based printed circuit board routing method. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 44(4):1420–1433, 2025.

[11] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023.

[12] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[13] Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. ChatEDA: A large language model powered autonomous agent for EDA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(10):3184–3197, 2024.

[14] Haoyuan Wu, Haisheng Zheng, Zhuolun He, and Bei Yu. Divergent thoughts toward one goal: LLM-based multi-agent collaboration system for electronic design automation, 2025. arXiv:2502.10857.

[15] Chia-Tung Ho, Haoxing Ren, and Brucek Khailany. VerilogCoder: Autonomous verilog coding agents with graph-based planning and abstract syntax tree (AST)-based waveform tracing tool. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 300–307, 2025.

[16] Lin Chen, Ran Chen, Shoubo Hu, Xufeng Yao, Zhentao Tang, Shixiong Kai, Siyuan Xu, Mingxuan Yuan, Jianye Hao, Bei Yu, and Jiang Xu. PCBAgent: An agent-based framework for high-density printed circuit board placement. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, pages 781–787, 2025.

[17] Jindong Li, Lianrong Chen, Bin Yang, Jiadong Zhu, Ying Wang, Yuzhe Ma, and Menglin Yang. PCB-Bench: Benchmarking LLMs for printed circuit board placement and routing. In *International Conference on Learning Representations*, 2026.

[18] Taiting Lu, Kaiyuan Lin, Ziwei Dong, Sisong Bei, Haolin Ye, Yuxin Tian, Runze Liu, Mingjia Wang, Jingying Zeng, Hongxing Pan, Kai Zhang, Haoyu Wang, Guoliang Shi, Ling Ma, Yifan Yang, Jiaying Lu, Qi He, Yi-Chao Chen, Sung-Liang Chen, Yincheng Jin, and Mahanth Gowda. OmniRouting: A semantic-coupled multimodal benchmark for constraint-aware spatial reasoning in PCB routing, 2026. arXiv:2608.04434.

[19] The OpenROAD Project. OpenROAD: Global and detailed routing. https://github.com/The-OpenROAD-Project/OpenROAD, 2026. Documentation for the FastRoute global router and TritonRoute detailed router.

[20] KiCad Development Team. KiCad 9.0 PCB editor documentation. Project documentation, 2026.

[21] Robert C. Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal*, 36(6):1389–1401, 1957.

[22] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[23] Brian Benchoff. OrthoRoute: A GPU-accelerated PCB autorouter for KiCad. Software repository, 2025.

[24] Andy Haas. KiCad Routing Tools. Software repository, 2026.

[25] OpenAI. Introducing GPT-5.4 mini and nano, 2026.

[26] Qwen Team. Qwen3.5-397B-A17B model card, 2026.

[27] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[28] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models, 2024.

[29] Anthropic. Introducing Claude Opus 5.5, 2026.

[30] Anthropic. Introducing Claude Haiku 5.5, 2026.

[31] Silicon Labs. AN0046: USB hardware design guidelines. Technical Report AN0046, Silicon Laboratories Inc., June 2020. Revision 1.02, Sections 3.1 and 3.3.1.

A. Overall Routing Algorithm

Algorithm 1 summarizes PCBRouteAgent. Initialization follows Sections 3.2–3.2.4; local recovery follows Section 3.3.2. The LLM call uses Equation 1 and the prompt templates in Appendix B.

Algorithm 1 PCBRouteAgent

Input: Initial PCB B_0, target nets $\mathcal{N}$, design rules $\mathcal{R}$, LLM $\mathcal{M}_\theta$, execution budget, maximum replanning rounds $r_{\max}$
Output: Saved PCB B_r and execution report

$(\mathcal{I}, S) \leftarrow \textsc{ExtractRepresentation}(B_0, \mathcal{N}, \mathcal{R})$
$\mathcal{G}_0 \leftarrow \textsc{InitializeGuide}(B_0, \mathcal{N}, \mathcal{R})$ ▷ MST, order, corridors, via regions
$B \leftarrow B_0$; $t \leftarrow 0$
while execution budget remains **do**
  $\mathcal{T} \leftarrow \textsc{CompilePendingTasks}(\mathcal{G}_t, B)$
  if $\mathcal{T} = \varnothing$ **then**
    break
  end if
  $\mathcal{D}_t \leftarrow \varnothing$
  for each connection task e in $\mathcal{T}$, in planned order **do**
    $C \leftarrow \textsc{Checkpoint}(B)$
    if checkpoint creation fails **then**
      go to finalization
    end if
    $(B', \mathcal{F}_e) \leftarrow \textsc{ExecuteAndRecover}(e, B, \mathcal{R})$
    if connectivity and board-state checks pass **then**
      $B \leftarrow B'$ ▷ Accept the connection
    else
      $B \leftarrow \textsc{Restore}(C)$; $\mathcal{D}_t \leftarrow \mathcal{D}_t \cup \{\mathcal{F}_e\}$
    end if
    if execution budget is exhausted or rollback fails **then**
      go to finalization
    end if
  end for
  if all target connections are established or $\mathcal{D}_t = \varnothing$ or $t \geq r_{\max}$ **then**
    break
  end if
  $\mathcal{K}_t \leftarrow \textsc{ConnectivitySummaries}(B, \mathcal{N})$
  $\Delta\mathcal{G}_t \leftarrow \mathcal{M}_\theta(\mathcal{I}, S, \mathcal{G}_t, \mathcal{D}_t, \mathcal{K}_t; P_{\text{sys}}, P_{\text{replan}})$
  Validate $\Delta\mathcal{G}_t$; request a bounded correction if needed
  $\mathcal{G}_{t+1} \leftarrow \mathcal{G}_t$
  if $\Delta\mathcal{G}_t$ is valid **then**
    $\mathcal{G}_{t+1} \leftarrow \textsc{ApplyPatch}(\mathcal{G}_t, \Delta\mathcal{G}_t)$ ▷ Preserve accepted routes
  end if
  $t \leftarrow t + 1$
end while
Finalization: $B_r \leftarrow \textsc{FinalizeAndSave}(B)$ ▷ Last consistent board state
Evaluate B_r using the official evaluator
return B_r and execution report

B. Prompt Templates

The following templates reproduce the fixed prompt wording used by the implementation. Brace-delimited placeholders denote runtime-expanded board data, view metadata, feedback, routing guides, JSON schemas, or examples. The Global planner templates are extracted from `route_agent/prompts/bundle.py`; the correction fields follow `route_agent/cli/pipeline.py`. The locally evaluated PCBWorld baseline uses the v5 templates in `methods/llm_agent/wrappers/prompts/v5.py` with S-expression board observations. Input images are attached separately to the Global planner requests.

B.1. Global Routing Planner Prompts

B.1.1. System Prompt

System Prompt for the Global Routing Planner

Prompt:

You are the Global routing planner for a printed circuit board.
You receive (1) board-context, per-layer copper-state and connectivity-demand images of the current committed board in one physical mm frame and (2) a
structured sidecar with exact ids and rules. You output a RouteGuidePlan as JSON only.

What a plan is:
- connection_trees: per net, edges that join two EXISTING connectivity clusters (ids like cluster:24:0). Never invent a terminal; Steiner regions are corridors, not clusters.
- corridors: axis-aligned regions per layer (bbox [x1,y1,x2,y2] in mm). They are SOFT: the Detailed router works inside them when it can and reports deviations. A region may cover obstacles; the Detailed router finds a legal path inside it.
- An edge lists its ordered stages (corridor ids). Consecutive same-layer stages must overlap. A layer change needs the two stages to overlap AND a via_region covering that overlap, listed in via_regions in order.
- Layer changes are YOUR decision only: the Detailed router never places a via and never leaves the layer of the stage it is in. A pad fenced in on its own layer (a dense SMD row, a pad walled by neighbouring pads or earlier tracks) cannot be left on that layer: start the edge on the other layer when the pad is thru-hole, otherwise put a small via_region (about 1 mm) right next to the pad between a short first stage on the pad's layer and the next stage on the other layer. Without that the edge is certain to fail.
- The first stage must contain an anchor of the source cluster on that layer; the last stage must contain an anchor of the target cluster.
- profile_ref must be one of the allowed profiles.
- net_order is your routing order; reservations are soft preferences.
- based_on.snapshot_id must copy the sidecar's snapshot_id exactly.

Plan only the work-set nets. Prefer short, wide corridors; use layer changes where a pad or a channel needs one and nowhere else; keep corridors inside the board bbox; use the images to avoid dense pad fields and existing copper. Do not draw copper. Output JSON matching the schema, nothing else.

B.1.2. Feedback-Driven Replanning Prompt

Prompt for Feedback-Driven Replanning

Images: Board-context, layer-specific copper-state, and connectivity-demand views.

Prompt:

Task
Produce a RouteGuidePlan for the work-set nets of this board.

Images
One frame, mm coordinates; pixel_to_mm affine per view is in the sidecar.views list.
- {view_id}: layers {view_layers}, bbox_mm {view_bbox_mm}, size {view_image_size}

Structured sidecar (JSON)
{structured_board_json}

Previous plan (revise only the affected items; keep the rest)
{current_routing_guide_json}

Detailed feedback (the failure you must fix)
{feedback_batch_and_live_state_json}

Return a GlobalGuidePatch JSON object: {patch_id, based_on: {snapshot_id, rules_hash, plan_revision: {plan_revision}}, replace_corridors, replace_via_regions, replace_edges, preserve, net_order?, budget_revision?}. Replace only what the feedback requires (a replaced edge or corridor is given in full); list every unaffected edge id in preserve. The cluster ids and snapshot_id stay those of the sidecar (the board was rolled back to that stable state plus the edges already done, see feedback.live_state). A full RouteGuidePlan is accepted instead of a patch but must then validate on its own.

GlobalGuidePatch JSON schema
{patch_json_schema}

Example patch (shape only; use the real ids, layers and mm coordinates of this board)
{patch_example_json}

If the feedback carries 'packets' (several edges, an end-of-pass batch), every edge in it failed in this pass and the board now holds everything else that was routed: return ONE patch that fixes as many of them as you can, those whose evidence makes the fix clear first (via_sites, head_is_thru, alt_starts), and list every edge you leave unchanged in preserve.

The Detailed never places a via on its own: a layer change exists only where the plan has a via region. When failure_evidence.via_sites is non-empty the head is fenced in on its layer; each entry is a clearance-checked [x, y] in mm right next to the stuck head where a via fits on both layers. To route the edge on the other layer, add a via region (a box about 1 mm wide around one of those points), a short first-stage corridor on the head's layer that contains the head and overlaps that box, and the next stage on the other layer overlapping it too. blocker_candidates lists the foreign pads / tracks fencing the head in (with positions and layers); head_is_thru true means the pad itself spans both layers, so a corridor starting on the other layer needs no via; alt_starts are other points of the net's copper near the first stage the edge could start from instead.

When a packet carries 'history', the edge has been re-planned before: each entry is the shape the Detailed executed in an earlier round (stages with layer and bbox, via regions) and where it died (died_at, head, reason). Those shapes have already failed: do not return one of them or a near copy. Change what failed: enter the target from its other side or on the other layer, move the via region away from died_at, or start the edge from one of alt_starts.

Rules the validator enforces: every corridor has id, layer and bbox; every via region has id, layers (the two layers it joins) and bbox; consecutive same-layer stages overlap; a layer change needs a via region

covering the overlap of the two stages; the first stage contains an anchor of source_cluster on its layer and the last stage an anchor of target_cluster on its layer.

Output Format: A JSON `GlobalGuidePatch`; the serialized schema and example are inserted at the indicated placeholders.

B.2. Guide-Patch Correction Prompt

The correction request reuses the system prompt and replanning user template above. Its feedback object additionally contains the following fields, including the exact correction instruction. The previous response and validation-error list are populated at runtime.

Additional Feedback Fields for Guide-Patch Correction

Prompt:

"previous_response": {previous_response_json},
"validation_errors": {validation_errors_json},
"instruction": "Your previous patch failed the checks listed in validation_errors. Return a corrected GlobalGuidePatch for the same edge(s); keep everything else unchanged."

Output Format: A corrected JSON `GlobalGuidePatch`.

B.3. PCBWorld Interactive Baseline Prompt

The system message contains the static board context; the user message supplies the current observation, action history, rejected-attempt information, and valid actions at each step.

System Prompt for the PCBWorld Interactive Baseline

Prompt:

You are a PCB routing agent for KiCad. Connect unconnected pins (points in routing_geometry) within each net. Never connect pins from different nets.

Priority
1. Complete all connections (reduce unconnected points to zero).
2. Avoid DRC violations (no crossing other nets, respect clearance).
3. Minimize total wirelength.

Routing Guidelines
- Targets are the (point ...) entries in routing_geometry; the final segment must land on the target pad's (x, y) while the cursor is on that pad's layer (points carry no layer — look up the pad in board_static or the nearby tracks).
- Use exact coordinates from the observation (pad positions, point targets, track endpoints). Only invent coordinates for detour waypoints.
- Avoid static obstacles, board edges, and existing tracks on the same layer (any net, including the current one).
- When a path is blocked, detour: switch layers with 'make_via' (offset $\geq$1mm from any pad — never place a via on a pad), route around on the opposite layer, then switch back.

- When the cursor is already near the remaining target with a clear path, prefer ‘finish ¡mode¿‘ over a manual ‘make_line‘.

Layer-change detour pattern
Route from (0, 0) on layer 1 to (10, 10) on layer 1, bypassing an obstacle via layer 2:
1. start_route 0.0 0.0 1
2. make_via 0.0 0.5 w # offset off the source pad, then cross to layer 2
3. start_route 0.0 0.5 2 # re-enter routing on the new layer
4. make_via 10.0 9.5 w # STOP before the target’s (x, y) and cross back
5. start_route 10.0 9.5 1 # re-enter routing on the target pad’s layer
6. make_line 10.0 10.0 w # final segment lands on the pad, on its own layer

Do NOT draw or end on a pad from the wrong layer
Never let ‘make_line‘ or ‘make_via‘ reach the target pad’s exact (x, y) while the cursor is on a different layer than that pad. If you do, you are trapped:
- placing a via at the pad violates the ”no via on pad” rule, and
- the pad is unreachable from the wrong layer, so ‘finish‘ will fail repeatedly.
Always make the layer-return ‘make_via‘ at an offset waypoint ($\geq$1mm away from the target pad) *before* the track arrives at the target’s (x, y).

Response Format
Output exactly one ¡think¿...¡/think¿ followed by one ¡action¿...¡/action¿. Never repeat think-action pairs.
”[no effect]” in history = action made no progress; ”Last rejected attempt” below = action was refused. Do not repeat either.

Board State
The board state is provided in S-expression format. All floating-point values are rounded to 3 decimal places. Each element is enclosed in parentheses: (tag value ...) or (tag (child ...) ...).

board_static — static board context (unchanged during episode)
(bbox x y w h) — board bounding box origin and size in mm
(scale N) — max(w, h), for normalization reference
(net_count N) — total number of nets
(copper_layers N) — number of copper layers
(boardlines (edge ID (p1 PID x y) (p2 PID x y)) ...) — board outline edges; an arc outline entry appears as (arc ID (p1 ...) (mid PID x y) (p2 ...)) with mid on the arc between p1 and p2 (full circle: p1 == p2, mid = opposite point)
(nets (net CODE ”NAME” (pads (pad ID x y layer) ...)) ...) — nets with pad center positions; layer is a copper layer number (1=Top, N=Bottom) or ‘th‘ for a through-hole pad (plated through every copper layer — start_route / make_via may land on any layer)
(obstacles (obs ID x y (size w h) (layer N)) ...) — static obstacles

routing_geometry — dynamic per-net routing state (updated every step)
(net CODE
(tracks (track ID (p1 PID x y) (p2 PID x y) (layer N)) ...) — placed track segments
(vias (via ID (center PID x y) (layers start end)) ...) — placed vias
(points (point ID x y) ...)) — unconnected ratsnest targets; layer info not provided (infer from the target pad or nearby tracks)

router_head — current router cursor state
(xy x y) — cursor position in mm
(layer N) — active copper layer (1=Top)
(net N) — currently selected net code (0=none)
(phase N) — 0=NET_SELECT, 1=START_ROUTE, 2=ROUTING
(is_routing true—false)

(routing_mode M) — m=MarkObstacles, p=PushAndShove, w=Walkaround
(step N ratio) — current step number and step/max_steps ratio
{board_static}

User Prompt for the PCBWorld Interactive Baseline

Prompt:

Step {current_step}
{dynamic_observation}

History (step {current_step}, {valid_step_count} valid actions taken)
{action_history}{rejected_attempts}

Valid Actions
Choose exactly one from:
{valid_actions}

Output Format: One `<think>...</think>` block followed by one `<action>...</action>` block, as specified in the system prompt.

C. Computational Cost and Efficiency

We compare PCBWorld's interactive agent and PCBRouteAgent using Claude Haiku 5.5 under the same local evaluation conditions. Table **6** reports per-board means for each test split. Time uses the same measurement scope for both methods. Model calls count requests to the language model, while input and output tokens and API charges are taken from the recorded usage. Clean-pass rates match the corresponding Haiku rows in the main routing-quality tables. All displayed values are rounded.

Table 6: Per-board routing success and computational cost with Claude Haiku 5.5 under the same evaluation conditions. Input and output tokens are in thousands (k). Within each split, bold and italic indicate the better and second-best values, respectively.

Set	Method	CP ↑	Time (s) ↓	Calls ↓	In. (k) ↓	Out. (k) ↓	Cost ($) ↓
D2	PCBWorld	*0.633*	*188.0*	*43.20*	*45.8*	*38.9*	*0.0240*
(128)	PCBRouteAgent	**1.000**	**0.8**	**0.05**	**1.6**	**0.1**	**0.0002**
D3-A	PCBWorld	*0.343*	*436.0*	*76.60*	*112.6*	*88.1*	*0.0553*
(99)	PCBRouteAgent	**0.929**	**27.0**	**0.87**	**19.9**	**5.8**	**0.0048**
D3-B	PCBWorld	*0.000*	*1817.0*	*263.90*	*900.0*	*352.0*	*0.2660*
(10)	PCBRouteAgent	**0.800**	**146.0**	**2.90**	**163.0**	**30.0**	**0.0330**
D3-C	PCBWorld	*0.000*	*4566.0*	*676.60*	*5430.0*	*816.0*	*0.9510*
(10)	PCBRouteAgent	**0.100**	**516.0**	**5.10**	**666.0**	**62.0**	**0.3930**

Efficiency relative to completed work. Table **7** relates resource use to completed connections and clean-pass boards. Tokens per connected edge divide the total input and output tokens by the number of connected target edges across the split. Cost per clean-pass board divides the total API cost of all evaluated boards, including failed boards, by the number of boards that pass. It is undefined when no board passes, denoted by a dash. This is an aggregate evaluation cost per success, not a measurement of repeated attempts on an individual board.

Table 7: Resource use normalized by completed work with Claude Haiku 5.5. Connected edges are reported as completed/required totals for each split. Lower token and API costs are better.

Set	Method	Connected edges	Tokens/edge ↓	$/clean-pass board ↓
D2	PCBWorld	*953/1096*	*11,374*	*0.0380*
	PCBRouteAgent	**1096/1096**	**202**	**0.0002**
D3-A	PCBWorld	*900/1301*	*22,079*	*0.1610*
	PCBRouteAgent	**1287/1301**	**1,972**	**0.0050**
D3-B	PCBWorld	*191/445*	*65,545*	—
	PCBRouteAgent	**443/445**	**4,355**	0.0410
D3-C	PCBWorld	*232/1178*	*269,324*	—
	PCBRouteAgent	**1080/1178**	**6,744**	3.9300

Aggregate resource use. Table **8** summarizes usage across all 247 boards. The cumulative time is the sum of per-board elapsed times, not the elapsed duration of a parallel batch. PCBRouteAgent reduces model calls by approximately 131×, combined input and output tokens by 8.8×, API cost by 4.3×, and cumulative time by 14×. It also requires fewer tokens per connected edge on every split (Table **7**). Large-board routing remains costly relative to successful completion: on D3-C, only one of ten PCBRouteAgent boards passes, resulting in an aggregate API cost of $3.93 per clean-pass board.

Table 8: Total computational cost across all 247 boards with Claude Haiku 5.5. Token counts are in millions (M); cumulative time sums the elapsed time of individual boards. Lower is better in every column.

Method	Calls	In. tokens (M)	Out. tokens (M)	API cost ($)	Time (h)
PCBWorld	*22,529*	*80.3*	*25.4*	*20.72*	*36.4*
PCBRouteAgent	**172**	**10.5**	**1.5**	**4.77**	**2.6**

D. Case Studies and Failure Analysis

D.1. Same-board comparison with PCBWorld

On a D3-C board with the same Claude Haiku 5.5 backbone, PCBRouteAgent completes 84 of 86 connections, compared with 10 for the PCBWorld baseline, an improvement of 86.0 percentage points (Figure **4**). Table **9** shows that this improvement also uses fewer model calls and tokens.

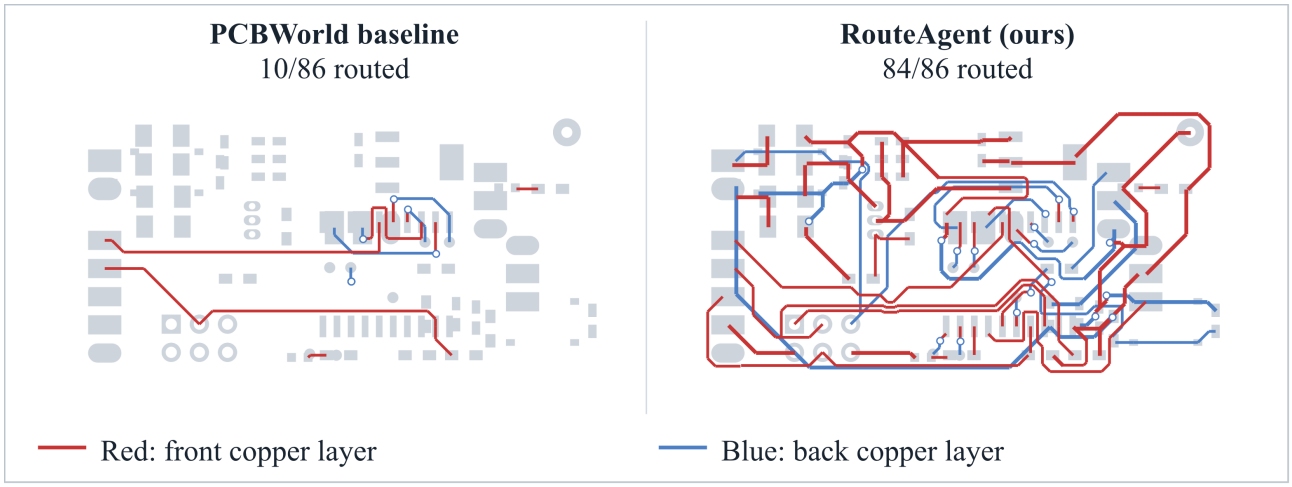


Figure 4: Saved routing outputs for the same D3-C board with Claude Haiku 5.5.

Table 9: Same-board routing quality and resource use with Claude Haiku 5.5.

Method	Pot. ↑	Rout. ↑	WL (mm) ↓	Via ↓	Model calls ↓	Input tokens ↓	Output tokens ↓
PCBWorld baseline	13.72	0.116	113.29	7	403	2,024,617	682,784
PCBRouteAgent (ours)	107.73	0.977	657.16	19	5	398,021	86,963

The PCBWorld baseline uses one model call per action and reaches the board's 403-step action limit. With five calls, PCBRouteAgent reduces recorded input and output tokens by 80.3% and 87.3%, respectively. This result is consistent with the planning–execution design: the model plans groups of connections, while the routing engine performs the individual path-search and copper-placement steps. When progress stalls, the model can revise a group of connections without selecting each low-level action separately. The comparison therefore illustrates how the framework can use the same model's decisions more efficiently while completing substantially more routing.

D.2. Failure analysis of incomplete routing

Earlier routing decisions can consume the local space needed by later connections. On the D3-C board in Figure **5**, the Opus variant completes 70 of 83 connections, whereas the reference completes all 83. Two signal nets must connect the controller and the sensor: SCL carries the clock signal and SDA carries data.

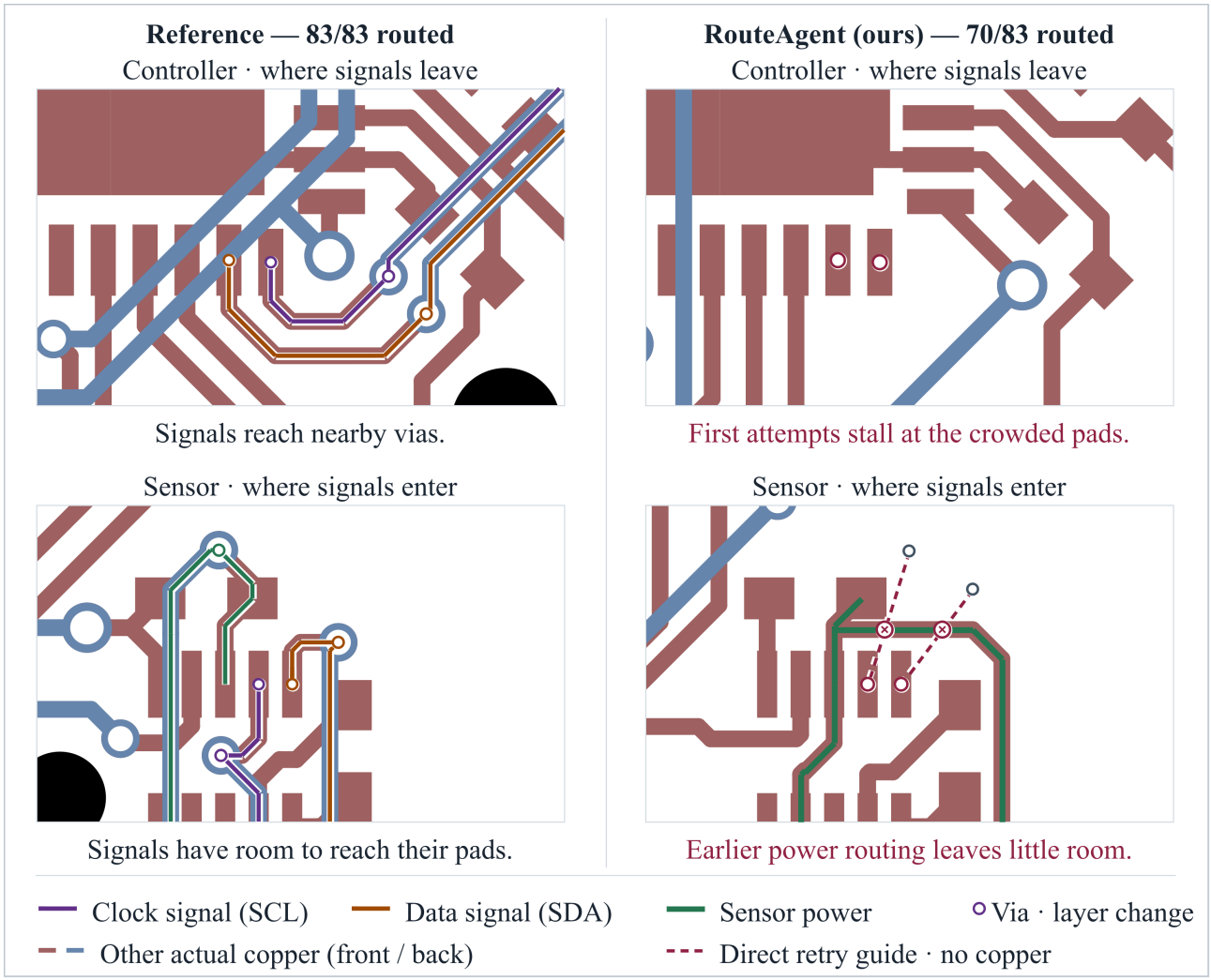


Figure 5: Controller and sensor regions in the reference (left) and PCBRouteAgent output (right). Dashed lines indicate retry guides, not laid copper.

The saved sequence routes sensor power before these signals, without reserving pad access or space for layer-changing vias. Initial attempts stall near crowded controller pads. Changing layers on retry gets the signals out, but their approaches to the sensor encounter the existing power route. Repair removes unfinished attempts while retaining that route, so the local restriction remains. The reference completes both signals with different paths and layer choices. This case suggests broader look-ahead in ordering and repair: reserve pad access, prioritize connections with few alternatives, and revisit earlier routes when needed.

D.3. Electrical constraints beyond routing completion

Full connectivity does not by itself establish electrical usability. On the D3-B board in Figure **3**, the Opus variant completes all 55 connections but routes USB D+ and D- independently.

The two USB traces carry data between the connector and controller as a differential pair [31], for which joint control of spacing and length balance is recommended. The reference traces run side by side and each measures 30.25 mm. PCBRouteAgent produces separate paths of 34.03 mm and 36.49 mm. The planning input specifies individual net widths and clearances but does not encode the pair relationship. This exposes a limitation in signal-specific planning: the task specification and checker need to capture constraints that depend on signal roles, in addition to connectivity and basic design rules. Electrical validation is still required to determine whether this full-speed USB link operates reliably.