CADET: CONSTRUCTION OF PCB ROUTING INSTANCES WITH EXECUTABLE DEMONSTRATIONS

Anonymous authors
Paper under double-blind review

ABSTRACT

Printed circuit board (PCB) routing is the stage of physical layout that finds paths connecting the placed pads of each net. Public datasets in this domain are few, and most contain only the final states of routed boards, not demonstrations of the electronic design automation (EDA) engine actions that produce them. To close this gap, we propose CADET (Constructing executAble Demonstrations for Engine-grounded Training), which jointly generates synthetic boards and demonstrations built from EDA engine actions. The resulting RouteDemos provides 100,000 synthetic boards routed with zero design rule violations (DRVs), together with their demonstrations. Their distributions over 17 statistics resemble those of 142 real reference boards. CADET trains a routing policy on these demonstrations with behavior cloning and fine-tunes it with reinforcement learning (RL). On real test boards, the policy raised the rate of complete DRV-free routing from 38.7% to 66.7% over an RL-only baseline and exceeded all other baselines in this rate, including open-source autorouters and GPT-6 Astra. We release the dataset and its generation code as open source.

1 INTRODUCTION

Printed circuit board (PCB) routing is the stage of physical layout that electrically connects the pads of each net under design rules. As pin counts and densities grow, manual routing becomes increasingly time-consuming, and its automation grows in importance (Yin et al., 2023). Rule-based autorouters still dominate in practice (Freerouting, 2026; Haas, 2026). Most learning-based routers either route cell by cell on a grid (Liao et al., 2020; Bonnet et al., 2024) or delegate path construction to an existing router (Zhou et al., 2023; Du et al., 2023). The former are limited by grid resolution, and the latter by the capability of the underlying router. PCBWorld (Song et al., 2026) sought to overcome these limitations by letting an agent directly use the gridless routing functions of a real electronic design automation (EDA) engine, but its performance still drops sharply on complex boards.

A key cause is the scarcity of training data for routing (Lin et al., 2022; Chen et al., 2025). First, routed boards are core intellectual property of individual companies and are rarely released. Public collections hold only about a thousand boards each (He et al., 2024; Lu et al., 2026). Second, in most cases, public boards contain only the completed board. The demonstration of how a designer operated the EDA tool is usually not recorded. Since many action sequences yield the same board, recovering the demonstration from a completed board is an ill-posed inverse problem. Finding an executable one is thus costly. Behavior cloning (BC) on action demonstrations followed by reinforcement learning (RL) fine-tuning has repeatedly succeeded across domains such as robot manipulation, games, and large language model (LLM) reasoning (Silver et al., 2016; Rajeswaran et al., 2018; Ouyang et al., 2022; Guo et al., 2025). Unfortunately, no public dataset yet provides routing demonstrations that execute in an EDA engine.

To fill this gap, we propose CADET (Constructing Executable Demonstrations for Engine-grounded Training). To our knowledge, CADET is the first method in EDA routing to synthesize executable routing demonstrations at scale. It generates boards whose statistics resemble those of a reference board dataset, together with engine-level action demonstrations that route each board completely with zero design rule violations (DRVs). The key idea is inductive construction. CADET adds pads one at a time and commits a pad only when an engine-level action sequence connects it to the existing

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

pads of its net with zero DRVs. Because each connection is added to an already routed board, the completed board is solvable by construction, and the recorded sequences form its demonstration. A best-first search tailored to routing finds these sequences efficiently. Finally, CADET re-executes every demonstration and keeps only those that again route the board completely with zero DRVs. As a result, CADET generates a large set of boards of comparable complexity from only a small reference board dataset. CADET then pretrains a routing policy on these demonstrations with BC and fine-tunes it with RL.

Our contributions are as follows.

Method. We propose CADET, which synthesizes routing boards with executable demonstrations through inductive construction in an EDA engine and trains a routing policy on them with BC followed by RL fine-tuning.

Data. We release RouteDemos, built with CADET, together with its generation code. It is the first large-scale dataset of routing demonstrations that execute in an EDA engine, with 100,000 synthetic routing instances. Its board statistics closely follow those of the reference board dataset.

Results. On real test boards, CADET raised the clean pass (CP) rate, the fraction of boards routed completely with zero DRVs, from 38.7% to 66.7% over RL alone on RouteDemos. CADET outperformed every baseline in CP, including open-source autorouters and GPT-6 Astra (OpenAI, 2026a).

2 PRELIMINARIES

2.1 PCB ROUTING

A PCB electrically connects component contacts, called *pads*, through copper traces. A *net* specifies pads that must be electrically connected. Let N be the number of nets, P_i the number of pads in net i, and $P = \sum_{i=1}^{N} P_i$ the total pad count. Pad p has location $\boldsymbol{x}_p$. Each net i is assigned a *netclass* from the board's netclass set $\mathcal{C}$. The netclass defines the net's rule vector $\boldsymbol{r}_i$, which includes track width w_i, clearance c_i, and via dimensions. Traces consist of track segments on copper layers, and *vias* connect tracks across layers. A board state s contains the board geometry, pads, net assignments, design rules, current routing, and the routing head, the position from which the next route segment starts. A routing instance is represented by its initial unrouted state s_0. Routing adds tracks and vias to connect every net while keeping distinct nets electrically isolated. A design rule check (DRC) reports DRVs of geometric constraints, such as minimum clearance. A successful solution completes all required connections with zero DRVs. Wirelength and via count characterize routing quality beyond this feasibility condition.

2.2 ROUTING MDP AND DEMONSTRATIONS

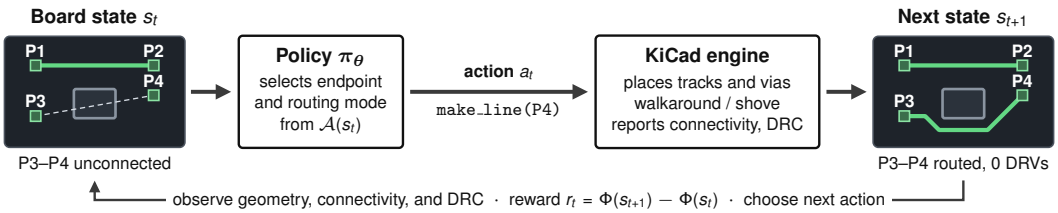


Figure 1: **Routing with an EDA engine.** At each step, the policy observes the board state s_t and selects an action $a_t \in \mathcal{A}(s_t)$ that invokes a KICAD operation. The engine executes it and returns the updated board s_{t+1} with connectivity and DRC results. In this example, the policy selects `make_line` toward P4, and the engine routes the track around the obstacle. The reward r_t is the resulting change in the potential Φ.

We use PCBWorld (Song et al., 2026), a Gymnasium environment (Towers et al., 2025) built on the KICAD EDA engine (Figure 1). At each step, the engine executes $a_t \sim \pi_\theta(\cdot \mid s_t)$ and produces $s_{t+1} = F(s_t, a_t)$, until completion or a step limit.

State. The state s_t is a nested dictionary of board objects with explicit coordinates, dimensions, layers, and net assignments. It contains fixed pads and design rules, current tracks and vias, connectivity, and the routing head. The environment provides s_t to LLM agents as S-expression text and to RL agents as indexed tokens.

Action. An engine-level action $a \in \mathcal{A}(s)$ specifies a native KiCad operation and its arguments, where $\mathcal{A}(s)$ is the candidate action set at state s. For example, `make_line`($\boldsymbol{x}_a$) extends a route toward a requested endpoint $\boldsymbol{x}_a$. The `make_via` operation additionally places a via to change layers. The policy selects a candidate endpoint and a routing mode, such as walkaround or push-and-shove. The engine constructs the resulting geometry (Appendix D lists all actions).

Reward. The environment defines board quality through a configurable potential,

$$\Phi(s) = \Phi_{\text{con}}(s) - \big[f_{\text{drv}}(s) + \lambda_w \ell(s) + \lambda_v n_{\text{via}}(s)\big], \qquad r_t = \Phi(s_{t+1}) - \Phi(s_t). \tag{1}$$

Here, f_{drv} penalizes engine-reported DRVs, ℓ is total wirelength, and n_{via} counts vias, with nonnegative weights λ_w, λ_v. The term Φ_{con} rewards connected and DRV-free nets and penalizes unrouted connections. Appendix F gives all terms, their weights, and a small penalty in r_t on invalid actions.

Demonstration. A demonstration is a trajectory $\tau = (s_0, a_0, s_1, \ldots, a_{T-1}, s_T)$ of engine-level actions and resulting states. Many trajectories can reach the same s_T, so a completed board does not determine its demonstration. For an action sequence τ, $F(s, \tau)$ denotes the state reached by executing τ in order from s. Replay reconstructs the states of τ by executing its actions from s_0. Fresh-engine replay does so on a newly started engine instance. Pads remain fixed throughout replay. Construction-time pad additions are outside this action space (§3.2).

3 Method

CADET trains a routing policy in the environment of §2.2. Instead of recovering demonstrations from completed boards, it constructs synthetic PCB instances and engine-executable demonstrations simultaneously. The policy learns from replayed demonstrations through BC, followed by RL fine-tuning on the same instances.

3.1 Board Attribute Distributions

The generator $\mathcal{G}_\psi$ constructs routable boards and demonstrations from sampled board attributes (Algorithm 1, lines 1, 2, and 5). These are pre-layout quantities that describe the board dimensions, netlist size, and design rules. Each attribute follows a simple parametric distribution or the empirical distribution of a reference dataset $\mathcal{D}_{\text{ref}}$. We denote these distributions collectively by q_ψ, with parameters ψ estimated from $\mathcal{D}_{\text{ref}}$. Appendix A.1 details these distributions and additional pad attributes.

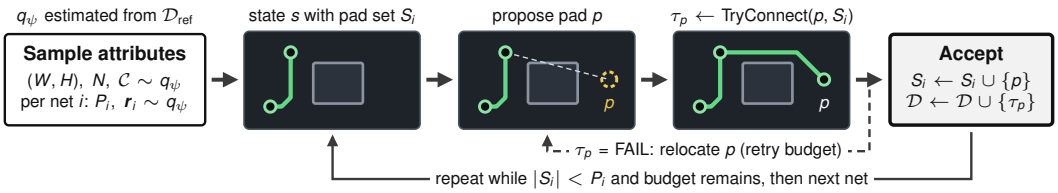
Board dimensions. Width W and height H define the board bounding box. The distribution $q_\psi(W, H)$ draws a board size $\sqrt{WH}$ and a long-to-short side ratio, which together determine W and H. The size follows the lognormal distribution $\sqrt{WH} \sim \text{LN}(\mu_{\text{size}}, \sigma^2_{\text{size}})$.

Netlist size. The net count N is the nearest integer to $\rho_{\text{net}} WH$, with the board area WH in cm^2, where the net density ρ_{net} follows $\text{LN}(\mu_{\text{net}}, \sigma^2_{\text{net}})$. This defines $q_\psi(N \mid W, H)$. The distribution $q_\psi(P_i)$ of each pad count $P_i \geq 2$ is the empirical distribution of per-net pad counts.

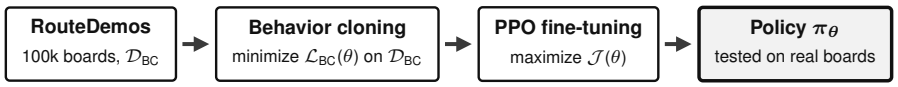
Design rules. The netclass set $\mathcal{C}$ and each rule vector $\boldsymbol{r}_i$ (§2.1) are sampled empirically from $q_\psi(\mathcal{C})$ and $q_\psi(\boldsymbol{r}_i \mid \mathcal{C})$, so all rule values in $\boldsymbol{r}_i$ are drawn jointly.

3.2 Board and Demonstration Generation

Inductive construction. For each net i, the generator starts from a pad set $S_i = \{p_0\}$ with p_0 at a uniformly chosen location and adds pads one at a time, each connected to the net by a route with zero DRVs (Algorithm 1, lines 6–12, and Figure 2a). Each subsequent pad is proposed near a randomly chosen existing pad of the net under a locality distribution (Appendix A.1). The construction state s holds all accepted pads, their routes, and the proposed pad. From s, TryConnect searches for an action sequence τ_p that connects the proposed pad to the net. Failed proposals are relocated within



(a) Board and demonstration generation



(b) Policy training

Figure 2: **Overview of CADET.** (a) Inductive construction. The generator samples board attributes from q_ψ and grows each net pad by pad. It accepts a proposed pad p only when TryConnect connects p to the pad set S_i with zero DRVs, and it adds the action sequence τ_p to $\mathcal{D}$. Every accepted pad thus leaves all retained nets connected with zero DRVs. A failed proposal is relocated within a retry budget. (b) The policy is trained by BC on the replayed demonstrations and then fine-tuned with PPO.

Algorithm 1 Constructing Routable Boards and Demonstrations

Sample $(W, H) \sim q_\psi(\cdot, \cdot)$ and $N \sim q_\psi(\cdot \mid W, H)$
Sample netclass set $\mathcal{C} \sim q_\psi(\cdot)$
Initialize demonstration set $\mathcal{D} \leftarrow \varnothing$ and pad sets $S_i \leftarrow \varnothing$ for all $i \in [N]$
for $i = 1, \ldots, N$ **do**
  Sample net i's attributes: $P_i \sim q_\psi(\cdot)$ and $\boldsymbol{r}_i \sim q_\psi(\cdot \mid \mathcal{C})$
  Propose initial pad p_0
  $S_i \leftarrow S_i \cup \{p_0\}$
  while $|S_i| < P_i$ and budget not exhausted **do**
    Propose pad p
    $\tau_p \leftarrow \text{TryConnect}(p, S_i)$
    if $\tau_p \neq$ FAIL **then**
      $S_i \leftarrow S_i \cup \{p\}$, $\mathcal{D} \leftarrow \mathcal{D} \cup \{\tau_p\}$
Form s_0 from the placed pads and set $\mathcal{D}_{\text{BC}} \leftarrow \text{Replay}(s_0, \mathcal{D})$ ▷ Algorithm 3

Algorithm 2 TryConnect(p, S_i): Best-First Connection Search

Require: Proposed pad p, pad set S_i of net i, current state s
Initialize queue $Q \leftarrow \{(\varnothing, a) : a \in \mathcal{A}(s)\}$
while $Q \neq \varnothing$ and budget not exhausted **do**
  Pop (τ_p, a) with the smallest $f(\tau_p, a)$ in Equation (2) from Q
  $\tau_p \leftarrow (\tau_p, a)$ and $s' \leftarrow F(s, \tau_p)$
  if s' has zero DRVs **then**
    if p is connected to S_i in s' **then**
      return τ_p
    else if the routing head of s' is unvisited **then**
      $Q \leftarrow Q \cup \{(\tau_p, a') : a' \in \mathcal{A}(s')\}$
return FAIL

a retry budget. Once the budget is exhausted, the net ends with fewer than the intended P_i pads. Because each pad is connected on an already routed board, the completed board is routable with high probability, and the sequences recorded in $\mathcal{D}$ form its candidate demonstration.

Connection search with TryConnect. TryConnect(p, S_i) searches for an executable action sequence τ_p that routes p to S_i from the current state s (Algorithm 2). Each action $a \in \mathcal{A}(s)$ routes from the routing head of s to its requested endpoint $\boldsymbol{x}_a$, which becomes the next routing head. Starting from p with an empty τ_p, the search appends actions one at a time. Each step selects the pair (τ_p, a) in the queue Q with the smallest estimated cost $f(\tau_p, a)$ in Equation (2), where a starts at

the routing head that τ_p reaches. It executes this pair and returns τ_p once the resulting state has zero DRVs and connects p to S_i (lines 3–6). Otherwise, a DRV-free state with an unvisited routing head adds (τ_p, a') to Q for each candidate a' from that head (line 9). It fails when the queue or expansion budget is exhausted.

The cost $f(\tau_p, a)$ combines the number of actions in τ_p with a grid-Dijkstra heuristic h that estimates the distance from the requested endpoint of a to S_i,

$$f(\tau_p, a) = 2(|\tau_p| + 1) + \mathbf{1}_{\text{via}}(a) + \begin{cases} 0, & \boldsymbol{x}_a \text{ lies on } S_i, \\ h(\boldsymbol{x}_a), & \text{otherwise}, \end{cases} \tag{2}$$

where $\mathbf{1}_{\text{via}}(a)$ adds a unit penalty if a is a via. Costs are estimated before execution in line 4.

After construction, we form s_0 from the placed pads of all nets and replay the action sequences in $\mathcal{D}$ in their recorded order to obtain the full trajectory τ. Replay keeps all retained pads fixed. Repair reroutes failed connections in τ, and pruning removes pads from s_0 that repair cannot connect (Algorithm 3). The pair (s_0, τ) is accepted only when fresh-engine replay executes every action, connects all nets, and yields zero DRVs. Otherwise the board is discarded. The accepted pair yields the state–action pairs $\mathcal{D}_{\text{BC}}$, written $\text{Replay}(s_0, \mathcal{D})$ in Algorithm 1 (line 13).

Pad-count calibration. Exhausted retry budgets and pruning can make the realized total pad count $\hat{P} = \sum_i \hat{P}_i$ fall below the sampled total $P = \sum_i P_i$. To compensate, the net count is increased to $\bar{N}$, the nearest integer to αN. The factor α is estimated in pilot runs by comparing the realized pad densities of generated boards with those of $\mathcal{D}_{\text{ref}}$ (Appendix A.1).

3.3 Behavior Cloning and RL Fine-Tuning

BC trains the policy on the demonstrations of all accepted boards, each verified by fresh-engine replay (Figure 2b). It minimizes the negative log-likelihood of the recorded engine-level actions over their state–action pairs $\mathcal{D}_{\text{BC}}$,

$$\mathcal{L}_{\text{BC}}(\theta) = -\mathbb{E}_{(s,a)\sim\mathcal{D}_{\text{BC}}} \log \pi_\theta(a \mid s). \tag{3}$$

BC learns from recorded decisions without optimal-action labels. Multiple routing sequences can solve the same board, and a successful demonstration need not maximize the routing reward.

The resulting policy $\pi_{\theta_{\text{BC}}}$ initializes PPO fine-tuning on the same boards. Fine-tuning maximizes the PPO clipped surrogate objective $\mathcal{J}_{\text{PPO}}$ (Schulman et al., 2017) with a KL penalty toward the BC policy,

$$\mathcal{J}(\theta) = \mathcal{J}_{\text{PPO}}(\theta) - \beta\, \mathbb{E}_s\big[D_{\text{KL}}\big(\pi_\theta(\cdot \mid s) \,\|\, \pi_{\theta_{\text{BC}}}(\cdot \mid s)\big)\big], \tag{4}$$

with $\beta = 0.05$ in the main configuration. Subsequent interaction allows the policy to improve under the routing objective.

We call the policy before fine-tuning CADET (BC). Scratch PPO uses random initialization on the same board dataset. BC does not train the critic, so fine-tuning begins with a critic warm-up. It also detaches the critic from the shared encoder. The comparison therefore evaluates the full BC and fine-tuning procedure, including its optimization settings. Generation and BC also incur costs beyond the RL interaction budget.

4 Experiments

4.1 Experimental Setup

Data. We evaluated routing policies on two real-board sets from the PCBWorld benchmark (Song et al., 2026), D3-A with 6 to 31 pads per board and D3-B with 31 to 100. We used 99 D3-A boards for evaluation only. From the 286 D3-B boards, we kept the 283 two-layer boards and split them into D3-B ref (142 boards) and D3-B test (141). CADET used D3-B ref as the reference dataset $\mathcal{D}_{\text{ref}}$ to produce RouteDemos, 100,000 boards with executable demonstrations, about 700 times the size of $\mathcal{D}_{\text{ref}}$. The PCBWorld synthetic training set D2 provides 10,000 boards with 8 to 21 pads each.

Baselines. The rule-based baselines were the open-source routers FreeRouting (Freerouting, 2026) and KiCad Routing Tools (KRT; Haas, 2026). The LLM baselines, GPT-6 Astra (OpenAI, 2026a)

and GPT-6 Luna (OpenAI, 2026b), routed boards by interacting with the PCBWorld environment (settings and prompts in Appendix J). PPO (X) denotes a policy trained from random initialization on dataset X. PPO (D2) reimplements the PCBWorld policy with our training configuration and exceeds the original in CP on D3-A. Table 2 labels it PCBWorld (PPO).

Evaluation. Every method was scored on its final KiCad board file with the same post-hoc DRC. CP is the fraction of boards with all connections completed and zero error-severity DRVs. Routability (Rout.) is the fraction of required connections completed. The potential gain $\Delta\Phi = \Phi(s_T) - \Phi(s_0)$ measures improvement over the unrouted board, with Φ from §2.2. FreeRouting and the learned policies ran four times per board.[1] Learned policies used a 1,024-step horizon. For any method, @k keeps the best of k rollouts in each run, by CP and then $\Delta\Phi$. Tables report means over these runs. Appendix C defines all metrics formally, and Appendix G adds standard deviations, wirelength, via counts, and DRVs. Time is the mean wall-clock time per board on a single server with an AMD EPYC 9354 CPU and one NVIDIA L40 GPU.

Training. All policies used the same six-layer Transformer and PPO hyperparameters. CADET applied BC and then 200 PPO fine-tuning iterations with $\beta = 0.05$, critic warm-up, and critic detachment. Scratch PPO ran 500 iterations. Appendix B lists the architecture, hyperparameters, and configuration differences. For each run, we selected the checkpoint with the highest mean $\Delta\Phi$ on D3-B ref. Appendix I provides the source code, software versions, compute, and baseline settings.

4.2 Analysis of RouteDemos

Acceptance rate and runtime. We analyzed the Algorithm 1 runs that produced RouteDemos. A pad that TryConnect fails to connect is relocated and, once the retry budget is exhausted, dropped from its net without discarding the board. Replay with repair and pruning (line 13) accepted 99.1% of the boards after fresh-engine replay. This high rate reflects that inductive construction keeps each board solvable before replay. The full pipeline takes about 146 hours on one 56-core CPU server. Appendix A.2 gives full details.

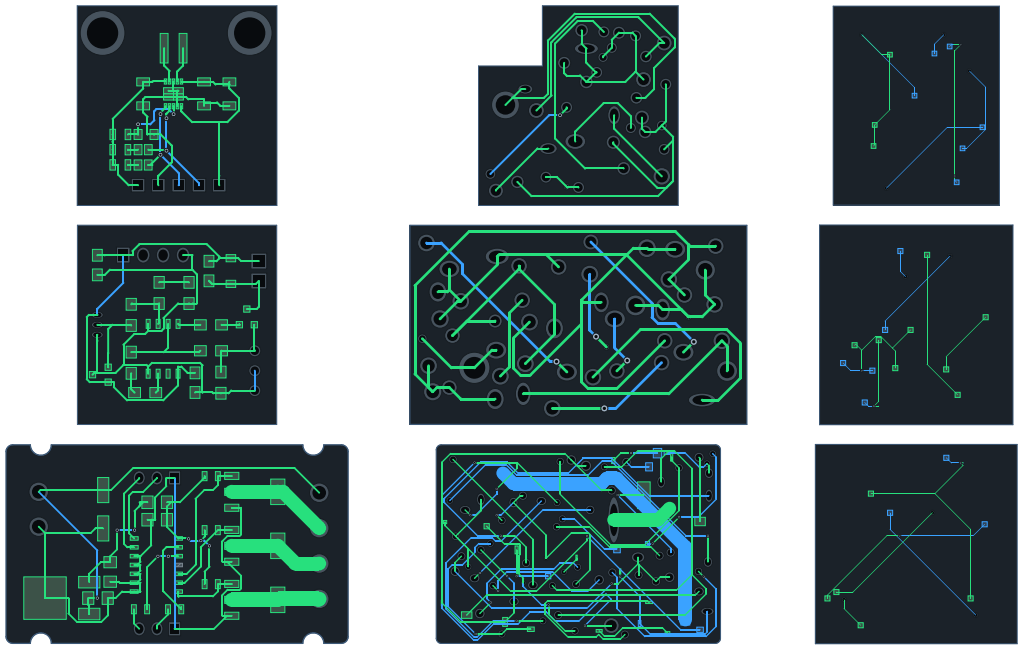
Table 1: Marginal KS distances to D3-B ref over 17 statistics, used to calibrate the generator. Lower is closer. Bold marks the lower distance between the two synthetic datasets. D3-B test provides a reference between real splits, not a lower bound. SMD and THT denote surface-mount and through-hole pads. Appendix A defines each statistic.

Statistic	D3-B test	RouteDemos	D2	Statistic	D3-B test	RouteDemos	D2
Board size	0.064	**0.075**	0.971	THT drill	0.108	**0.058**	0.970
Aspect ratio	0.170	**0.072**	0.504	Clearance	0.045	**0.018**	0.933
Net density	0.071	**0.107**	1.000	Track width	0.059	**0.045**	0.630
Pad density	0.057	**0.139**	1.000	Netclasses	0.028	**0.129**	0.660
Locality	0.026	**0.066**	0.392	THT fraction	0.057	**0.026**	0.444
Pads per net	0.015	**0.016**	0.108	Front SMD fraction	0.044	**0.037**	0.728
SMD short side	0.078	**0.024**	0.985	Nets per board	0.050	**0.236**	0.986
SMD long side	0.087	**0.019**	0.907	Pads per board	0.021	**0.245**	1.000
THT pad size	0.106	**0.149**	0.917	Mean of all	0.064	**0.086**	0.773

Quantitative similarity. Table 1 reports two-sample Kolmogorov–Smirnov (KS) distances (Smirnov, 1948) to D3-B ref, used for generator calibration. RouteDemos has lower KS distances than D2 on all 17 statistics, and its mean KS distance of 0.086 is 89% lower than the 0.773 for D2 and close to the 0.064 for D3-B test. Appendix A reports all 17 distances to D3-B test and the sample sizes.

Qualitative similarity. Figure 3 shows example boards from each dataset. RouteDemos not only provides engine-level demonstrations but also contains boards much closer to real boards than D2 in shape and complexity. D2 boards are smaller and sparser. RouteDemos samples design rules empirically from reference boards and thus reproduces their diverse track widths.

[1] KRT is deterministic and LLM inference is costly, so each ran once.



(a) D3-B test (b) RouteDemos (c) D2

Figure 3: Example routed boards from D3-B test, RouteDemos, and D2. Green and blue traces lie on different copper layers.

4.3 Main Results

Table 2: Evaluation on real test boards. Reference is the original human layout. Bold marks the best value among the other methods. CP and Rout. are in percent, and Time is in seconds per board.

		D3-B test (141)				D3-A (99)			
Method	Type	CP	$\Delta\Phi$	Rout.	Time	CP	$\Delta\Phi$	Rout.	Time
Reference	–	100.0	144.41	100.0	–	100.0	50.61	100.0	–
FreeRouting @1	Rule-based	65.6	119.33	**96.8**	45.6	82.1	46.05	97.7	8.6
KRT @1	Rule-based	27.7	84.71	78.3	**2.7**	69.7	40.54	92.8	**1.5**
GPT-6 Astra @1	LLM	56.0	104.20	86.0	491.7	83.8	45.03	93.8	138.9
GPT-6 Luna @1	LLM	5.0	37.72	47.9	335.7	46.5	31.30	78.5	179.1
PCBWorld (PPO) @1	RL	37.4	78.81	71.9	–	75.3	40.93	87.0	–
Cadet @1	BC+RL	66.7	107.60	90.7	14.1	91.4	45.17	97.5	3.8
Cadet @3	BC+RL	**80.3**	**120.96**	95.4	43.4	**97.5**	**48.65**	**99.3**	11.6

Cadet @1 outperformed every baseline in CP on both evaluation sets (Table 2). This result shows the effectiveness of BC on engine-level demonstrations that guarantee a clean pass. Cadet @1 also exceeded every other agent in the same environment on every metric. The 6.79M-parameter Cadet @1 outperformed the frontier GPT-6 Astra and ran about 35 times faster. Its margin over the lower-priced GPT-6 Luna was far wider. PCBWorld (PPO) uses the same architecture without BC. Cadet @1 exceeded it by 29.3% in CP and by 28.79 in $\Delta\Phi$ on D3-B test, whose boards have more pads, and by 16.1% in CP on D3-A.

Among the rule-based routers, KRT ran fastest but reached only 27.7% CP on D3-B test. Cadet @1 exceeded both KRT and FreeRouting in CP, although FreeRouting retained higher routability and $\Delta\Phi$ on D3-B test. With a wall-clock time comparable to that of FreeRouting on D3-B test (43.4 versus 45.6 seconds), Cadet @3 outperformed it by 14.7% in CP. On D3-A, Cadet @3 nearly matched the reference, with 97.5% CP and 99.3% routability.

4.4 WHERE DOES THE GAIN COME FROM?

Table 3: Effect of demonstrations and training data. CADET is CADET @1 of Table 2. Bold marks the best value in each column. CP and Rout. are in percent.

	D3-B test (141)			D3-A (99)		
Method	CP	$\Delta\Phi$	Rout.	CP	$\Delta\Phi$	Rout.
CADET	**66.7**	**107.60**	90.7	**91.4**	**45.17**	**97.5**
CADET (BC)	49.3	82.00	83.8	83.3	41.81	94.4
PPO (RouteDemos)	38.7	95.86	**92.2**	79.5	44.77	96.9
PPO (RouteDemos w/o TryConnect)	34.0	93.15	91.9	76.3	43.72	96.7
PPO (D2)	37.4	78.81	71.9	75.3	40.93	87.0
PPO (D3-B ref)	33.2	91.31	91.1	74.0	42.20	95.2

Table 3 traces the gain of CADET over PPO (D2) in two steps, changing the training boards and then adding demonstrations.

Effect of training boards. Among the PPO policies, PPO (RouteDemos) achieved the best value on every metric. On D3-B test, it matched PPO (D2) in CP and exceeded it by 17.05 in $\Delta\Phi$ and by 20.3% in routability. It also exceeded PPO (D3-B ref). D3-B ref contains complex boards designed by engineers, but its 142 boards, versus 100,000 in RouteDemos, show how the scarcity of real boards limits training. The variant RouteDemos w/o TryConnect only places pads, without connecting them by TryConnect, so its boards are not guaranteed to admit a clean pass. PPO (RouteDemos w/o TryConnect) was below PPO (RouteDemos) on every metric. Because TryConnect also determines which pads are accepted, this comparison assesses the construction procedure jointly.

Effect of demonstrations. Demonstrations had the dominant effect on CP. Although both used the same training boards, CADET exceeded PPO (RouteDemos) on D3-B test by 28.0% in CP (66.7% versus 38.7%) and by 11.74 in $\Delta\Phi$. This result shows that, as in other sequential decision problems (Rajeswaran et al., 2018), executable demonstrations, which public boards lack, make BC effective for engine-level routing, beyond what synthetic boards alone provide. The comparison covers the full BC and fine-tuning procedure, not initialization alone. Both CADET and CADET (BC) had lower D3-B test routability than PPO (RouteDemos). We attribute this to demonstrations that all end in a clean pass. The policy keeps working on a difficult net instead of abandoning it, which raises CP but can leave other nets unrouted. Fine-tuning improved every metric over CADET (BC) on both evaluation sets. Offline RL on demonstrations that include some unclean ones is a promising direction for reducing this trade-off.

5 RELATED WORK

Rule-based routers. PCB autorouters such as FreeRouting (Freerouting, 2026) and KRT (Haas, 2026) combine path search with rip-up and rerouting. They remain the practical standard but have difficulty completing complex, dense boards. Integrated circuit (IC) routers (Liu et al., 2020; Kahng et al., 2021) operate under different layer counts, design rules, and routing primitives.

Learning-based routing. Learned routers act at the grid, router, or engine level. Grid-level policies move a route one cell at a time, learning from reward (Liao et al., 2020; He et al., 2022; Bonnet et al., 2024) or from maze-router and manual routes (He & Bao, 2020). Their action set is small, but real boards need a fine grid resolution, which makes the search impractically large (Song et al., 2026). Router-level policies delegate path construction to a rule-based router and choose only some of its inputs. These inputs include net orderings (Lin et al., 2022; Zhou et al., 2023; 2025; Liao et al., 2026), router cost weights (Khan & Rovinski, 2026), fanout endpoints (Li et al., 2023), and coarse routing guidance (Zhu et al., 2019; Du et al., 2023; Liu et al., 2024; Chen et al., 2025; Shi et al., 2025). Some learn from reward and others from routes produced by people or solvers. The policy cannot correct paths that the router fails to build, so performance depends heavily on the underlying router. Engine-level policies act through the routing operations of an EDA engine. PCBWorld (Song et al., 2026) exposes native KICAD routing operations as actions. This intermediate level lets the policy shape route geometry with far fewer steps than grid moves. However, current engine-level

policies remain limited on complex boards. To our knowledge, no prior work trains an engine-level routing policy from demonstrations.

Routing datasets. PCBench (He et al., 2024) and OmniRouting (Lu et al., 2026) collect real designs, and PCBWorld (Song et al., 2026) builds its real-board benchmarks from PCBench. These resources hold at most completed boards, not demonstrations. Synthetic generators from PCBWorld and Jumanji provide unrouted instances (Bonnet et al., 2024; Song et al., 2026). Grid generators lack a feasibility guarantee (Liao et al., 2020), store only final Steiner trees (Lee et al., 2022), or discard the solving walks (Bonnet et al., 2024). To our knowledge, no public real or synthetic dataset stores routing actions that replay in an EDA engine.

6 CONCLUSION

We proposed CADET to address a key bottleneck of learning-based routing, the scarcity of demonstrations that execute in an EDA engine. CADET generates engine-level demonstrations and trains a routing policy on them with BC followed by RL fine-tuning. Its inductive construction adds pads one at a time, each with an engine-level action sequence that connects it to its net. By maintaining zero DRVs at every step, it efficiently produces complex DRV-free boards with the desired statistics. CADET estimates the distributions of key board attributes from real reference boards, and the resulting RouteDemos closely matches these boards in 17 statistics. On real test boards, the trained policy exceeded every baseline in CP and raised it from 38.7% to 66.7% over RL alone on the same boards. Our analysis shows the potential of DRV-free executable demonstrations for learned EDA layout. It also motivates more refined demonstration generation and the collection of expert demonstrations from human engineers.

AI USE STATEMENT

We used LLMs (Claude Opus 5.5 and GPT-6 Astra) to write code, draft and polish sections of the paper, search for related work, and discuss research ideas. The authors reviewed all generated code, checked every cited work, and revised all text. The GPT-6 Astra and GPT-6 Luna agents in §4.1 are baselines evaluated through their APIs. We take responsibility for the final content of this work, including text, claims, and artifacts.

ETHICS STATEMENT

This work involves no human subjects or personal data. RouteDemos contains only synthetic boards, and the release excludes the real D3-B boards (Appendix K).

REPRODUCIBILITY STATEMENT

Anonymized source code for generation, training, and evaluation is available at `https://anonymous.4open.science/r/CADET-FBBF`. §3.2 and Algorithms 1, 2, and 3 specify the construction of RouteDemos, and Appendix A.1 gives the fitted distributions and pad-count calibration. §4.1 defines the data splits and evaluation protocol, and Appendices B and C give the training settings and metrics. Table 10 in Appendix I maps every component, including software and compute, to its specification. Appendix K documents the released dataset.

REFERENCES

Clem Bonnet, Daniel Luo, Donal Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries Smit, Nathan Grinsztajn, Raphael Boige, Cemlyn Waters, Mohamed Ali Mimouni, Ulrich Mbou Sob, Ruan de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnu Pretorius, and Alexandre Laterre. Jumanji: a diverse suite of scalable reinforcement learning environments in JAX. In B. Kim, Y. Yue, S. Chaudhuri, K. Fragkiadaki, M. Khan, and Y. Sun (eds.), *International Conference on Learning Representations*, volume 2024, pp. 49264–49293, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/file/d6c38e81801a8d10afeabdb0a32fca8a-Paper-Conference.pdf.

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. ISSN 1937-4151. doi: 10.1109/tcad.2024.3486241. URL http://dx.doi.org/10.1109/TCAD.2024.3486241.

Xingbo Du, Chonghua Wang, Ruizhe Zhong, and Junchi Yan. HubRouter: Learning global routing via hub generation and pin-hub connection. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 36, pp. 78558–78579, 2023. doi: 10.52202/075280-3435.

Freerouting. Freerouting: Advanced PCB Auto-Router, 2026. URL https://github.com/freerouting/freerouting.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64 (12):86–92, 2021. doi: 10.1145/3458723.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi: 10.1038/s41586-025-09422-z.

Andy Haas. KiCad Routing Tools, 2026. URL https://github.com/drandyhaas/KiCadRoutingTools.

Youbiao He and Forrest Sheng Bao. Circuit Routing Using Monte Carlo Tree Search and Deep Neural Networks, 2020. URL https://arxiv.org/abs/2006.13607.

Youbiao He, Hebi Li, Tian Jin, and Forrest Sheng Bao. Circuit routing using Monte Carlo tree search and deep reinforcement learning. In *2022 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pp. 1–5, 2022. doi: 10.1109/VLSI-DAT54769.2022.9768074.

Youbiao He, Jacob Frieden, Hebi Li, Roba Abbajabal, Ge Luo, and Forrest Sheng Bao. PCBench: A Dataset for Printed Circuit Board Routing. DAC Work-in-Progress Poster, 2024. URL https://61dac.conference-program.com/presentation/?id=RESEARCH617&sess=sess237.

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. doi: 10.1109/TCAD.2020.3003234.

Afsara Khan and Austin Rovinski. Accelerating detailed routing convergence through offline reinforcement learning. In *Proceedings of the 2026 Design, Automation & Test in Europe Conference (DATE)*, pp. 1–7, 2026. doi: 10.23919/DATE69613.2026.11539594.

Kanghoon Lee, Youngjoon Park, Han-Seul Jeong, Deunsol Yoon, Sunghoon Hong, Sungryull Sohn, Minu Kim, Hanbum Ko, Moontae Lee, Honglak Lee, Kyunghoon Kim, Euihyuk Kim, Seonggeon Cho, Jaesang Min, and Woohyung Lim. ReSPack: A large-scale rectilinear Steiner tree packing data generator and benchmark. In *NeurIPS 2022 Workshop on Synthetic Data for Empowering ML Research*, 2022. URL https://openreview.net/forum?id=TpYf4EumAi.

Haiyun Li, Jixin Zhang, Ning Xu, and Mingyu Liu. FanoutNet: A Neuralized PCB Fanout Automation Method Using Deep Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(7):8554–8561, 2023. ISSN 2159-5399. doi: 10.1609/aaai.v37i7.26030. URL http://dx.doi.org/10.1609/aaai.v37i7.26030.

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. doi: 10.1115/1.4045044.

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. doi: 10.1016/j.eswa.2026.131424.

Yibo Lin, Tong Qu, Zongqing Lu, Yajuan Su, and Yayi Wei. Asynchronous reinforcement learning framework and knowledge transfer for net-order exploration in detailed routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(9):3132–3142, 2022. doi: 10.1109/TCAD.2021.3117505.

Jinwei Liu, Chak-Wa Pui, Fangzhou Wang, and Evangeline F. Y. Young. CUGR: Detailed-routability-driven 3D global routing with probabilistic resource model. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2020. doi: 10.1109/DAC18072.2020.9218646.

Ruizhi Liu, Zhisheng Zeng, Shizhe Ding, Jingyan Sui, Xingquan Li, and Dongbo Bu. NeuralSteiner: Learning Steiner tree for overflow-avoiding global routing in chip design. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 37, pp. 127346–127368, 2024. doi: 10.52202/079017-4045.

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. arXiv preprint arXiv:2608.04434, 2026. URL https://arxiv.org/abs/2608.04434.

OpenAI. GPT-6 Astra: A new generation of intelligence, 2026a. URL https://openai.com/index/gpt-6-astra/.

OpenAI. Introducing GPT-6 Sol and Luna, 2026b. URL https://openai.com/index/introducing-gpt-6-sol-and-luna/.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 35, pp. 27730–27744, 2022. doi: 10.52202/068431-2011.

Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Robotics: Science and Systems XIV*, RSS2018. Robotics: Science and Systems Foundation, 2018. doi: 10.15607/rss.2018.xiv.049. URL http://dx.doi.org/10.15607/RSS.2018.XIV.049.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal Policy Optimization Algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Liangliang Shi, Shenhui Zhang, Xingbo Du, Nianzu Yang, and Junchi Yan. DSBRouter: End-to-end global routing via diffusion Schrödinger bridge. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, pp. 55065–55081, 2025.

David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016. doi: 10.1038/nature16961.

N. Smirnov. Table for estimating the goodness of fit of empirical distributions. *The Annals of Mathematical Statistics*, 19(2):279–281, 1948. doi: 10.1214/aoms/1177730256.

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, 2026. URL https://arxiv.org/abs/2607.05915.

Mark Towers, Ariel Kwiatkowski, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Jordan K. Terry, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 38, pp. 163114–163129, 2025. doi: 10.52202/085713-4916.

Shiyuan Yin, Min Jin, Gang Chen, Guoliang Gong, Wenyu Mao, and Huaxiang Lu. Unet-Astar: A deep learning-based fast routing algorithm for unified PCB routing. *IEEE Access*, 11:113712–113725, 2023. doi: 10.1109/ACCESS.2023.3323589.

Zhanwen Zhou, Hankz Hankui Zhuo, Xiaowu Zhang, and Qiyuan Deng. XRoute Environment: A novel reinforcement learning environment for routing. arXiv preprint arXiv:2305.13823, 2023. URL https://arxiv.org/abs/2305.13823.

Zhanwen Zhou, Hankz Hankui Zhuo, Jinghua Zhou, and Wushao Wen. Transformer-based reinforcement learning for net ordering in detailed routing. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 9492–9500, 2025. doi: 10.24963/ijcai.2025/1055.

Keren Zhu, Mingjie Liu, Yibo Lin, Biying Xu, Shaolan Li, Xiyuan Tang, Nan Sun, and David Z. Pan. GeniusRoute: A new analog routing paradigm using generative neural network guidance. In *Proceedings of the 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019. doi: 10.1109/ICCAD45719.2019.8942164.

A GENERATED-BOARD STATISTICS

Table 4 reports all 17 marginal KS distances to D3-B test boards. It complements the D3-B ref comparison in Table 1. The Kolmogorov–Smirnov (KS) distance is the maximum absolute difference between empirical cumulative distribution functions. Smaller values indicate closer distributions.

The statistical analysis included 142 D3-B ref boards, 141 D3-B test boards, and 10,000 D2 boards. For RouteDemos, an area filter retained 9,994 boards from a random sample of 10,000.

Table 4: Full marginal KS distances to D3-B test. The D3-B ref column measures variation between the two real-board splits. It is a reference value, not an achievable lower bound for a generator. SMD and THT denote surface-mount and through-hole pads, respectively.

Statistic	D3-B ref	RouteDemos	D2
Board size, $\sqrt{WH}$ (mm)	0.064	0.102	0.972
Aspect ratio, $\ln(\text{long}/\text{short})$	0.170	0.124	0.606
Net density (nets/cm^2)	0.071	0.154	1.000
Pad density (pads/cm^2)	0.057	0.167	1.000
Locality, minimum spanning tree (MST) edge length / board diagonal	0.026	0.078	0.369
Pads per net	0.015	0.014	0.101
SMD pad short side (mm)	0.078	0.093	0.979
SMD pad long side (mm)	0.087	0.094	0.835
THT pad size (mm)	0.106	0.233	0.910
THT drill diameter (mm)	0.108	0.116	0.970
Netclass clearance (mm)	0.045	0.057	0.919
Netclass track width (mm)	0.059	0.031	0.599
Netclasses per board	0.028	0.108	0.645
THT fraction of pads	0.057	0.059	0.429
Front-side fraction of SMD pads	0.044	0.029	0.714
Nets per board	0.050	0.207	1.000
Pads per board	0.021	0.245	1.000
Mean of all	0.064	0.112	0.768

RouteDemos has smaller KS distances than D2 on every measured axis. Its distances range from 0.014 for pins per net to 0.245 for pads per board. Distances for aspect ratio, pins per net, track width, and the front-side fraction of SMD pads are below the corresponding D3-B ref–test references. These distances compare marginal distributions.

Table 5: Selected proportions. Proportions are fractions, not percentages.

Statistic	D3-B test	D3-B ref	RouteDemos	D2
All-THT boards	0.262	0.291	0.297	0.000
All-front SMD	0.615	0.620	0.621	0.005
Two-pad nets	0.610	0.600	0.616	0.601
Three-pad nets	0.194	0.195	0.193	0.199

Table 5 provides complementary summaries. RouteDemos approximates several discrete proportions, including through-hole-only boards and two- and three-pad nets.

A.1 GENERATION DETAILS

Sampling distributions. Sampling statistics use the 142 D3-B ref boards. The fitted size variable is $\sqrt{WH}$, where W, H are the board bounding-box dimensions in millimeters. Its lognormal distribution has median 39.95 mm and log standard deviation 0.458. Samples are clipped to $[10, 105]$ mm. The log aspect ratio $\log(\max(W, H)/\min(W, H))$ has a point mass of 0.128 at zero. Otherwise, it follows a half-normal distribution with scale 0.655. The exponentiated draw is the long-to-short side ratio. It is capped at 4.3 and subsequently at $(\sqrt{WH}/12)^2$ before conversion to side lengths. Outlines are sampled from a fixed mixture of rectangles, rounded rectangles, polygons, and circles with probabilities 0.53, 0.25, 0.20, and 0.02. These shape probabilities are not fitted.

Table 6: D3-B ref sampling models before connection acceptance.

Statistic	Sampling model
Size $\sqrt{WH}$, net density	Lognormal
Log aspect ratio	Point mass and half-normal
Netclass sets, layer fractions, P_i	Empirical resampling
Pad dimensions and drill	Empirical, conditional on type
Same-net MST edge length$/d$	Lognormal locality

Net density ρ_{net}, in nets/cm^2, follows a lognormal distribution with median 1.032 and log standard deviation 0.902. Size and density are sampled independently before connection acceptance. The calibrated scale α increases the net count to $\bar{N}$ through

$$\bar{N} = \mathrm{clip}_{[2,41]}\left(\mathrm{round}\left(\alpha\rho_{\mathrm{net}}\frac{WH}{100}\right)\right), \qquad P = \sum_{i=1}^{\bar{N}} P_i. \tag{5}$$

Each P_i is independently resampled from 2,506 observed net pad counts filtered to $[2, 64]$. We sample one real board's complete netclass set and assign a class to each generated net using its observed class weights. The through-hole fraction and front-side surface-mount fraction are jointly resampled as one board-level pair. Pad type and surface-mount layer are then sampled using these fractions. Dimensions and drill diameters are resampled conditional on pad type, with width and height exchanged with probability 0.5.

For locality, a lognormal distribution is fitted to same-net MST edge lengths divided by the board diagonal. Its median is 0.1177 and log standard deviation is 0.90. This distribution proposes the second and subsequent pads around a uniformly selected accepted pad of the same net. The first pad of each net is proposed uniformly over the bounding box. Accepted distances need not retain the fitted distribution because site checks and connection search reject proposals.

With $d = \sqrt{W^2 + H^2}$ and an accepted same-net anchor pad at $\boldsymbol{x}_{\mathrm{anc}}$, a subsequent pad p is proposed at

$$\boldsymbol{x}_p^{\mathrm{prop}} = \boldsymbol{x}_{\mathrm{anc}} + du(\cos\varphi, \sin\varphi)^\top, \quad u \sim \mathrm{LN}(\mu_{\mathrm{loc}}, \sigma_{\mathrm{loc}}^2), \quad \varphi \sim \mathcal{U}[0, 2\pi). \tag{6}$$

Here μ_{loc} and σ_{loc} parameterize the lognormal locality distribution. The site is snapped to the placement grid before geometric checks.

Construction gates and budgets. Site sampling uses a 0.1 mm grid and at most 60 retries per site-sampling call. The generator allows up to 30 connection attempts for a pad. The first pad is provisional, and the first pair is committed only after a successful connection. Failure to find a site or exhaustion of connection attempts stops the current net, including when its first or second pad fails. A net with fewer than two committed pads is discarded. A 1,500 s construction limit is checked after each net and does not include finalization.

TryConnect loads the proposed board while retaining accepted routing. It rejects initial geometric DRVs and a new pad already connected without routing, selects its net, and searches for a connection. A post-connection check rejects remaining required connections and strict DRC failures, including dangling tracks. Successful proposals retain their pads, routing, and actions. Failed proposals leave the retained construction unchanged. Consequently, connectivity and zero evaluated geometric DRVs hold at accepted extensions, not necessarily at provisional sites or intermediate search states.

Connection search. Each queue entry (τ_p, a) stores the engine checkpoint, wrapper snapshot, observation, and routing head reached by τ_p. Candidate actions comprise direct requests to up to three nearby same-layer targets, eight directional requests at each of four scales (0.2, 1, 5, and 25 mm), and a via to another layer. Diagonal requests multiply these scales by $\sqrt{2}$, so 25 mm is not a maximum hop distance. Each recorded hop pairs route initialization with a line or via operation. During search, initialization is executed only when the routing session is closed. The count $|\tau_p|$ in Equation (2) counts the actions in τ_p, not wirelength.

The construction cost in Equation (2) uses walkaround mode. During final repair, candidates with $\boldsymbol{x}_a$ on S_i also use push-and-shove and mark-obstacles modes. Their costs are

Algorithm 3 Finalization: Replay, Repair, and Pruning

Require: Unrouted instance s_0, action sequences $\tau_p \in \mathcal{D}$ in recorded order
for $k = 1, \ldots, 16$ **do**
  $s \leftarrow s_0$
  for each $\tau_p \in \mathcal{D}$ in recorded order **do**
    if replay of τ_p from s fails **then**
      $\tau_p \leftarrow \mathrm{TryConnect}(p, S_i)$ from s with the cost in Equation (7) ▷ repair
      if $\tau_p = \mathrm{FAIL}$ **then**
        Prune p and continue with the next k
    $s \leftarrow F(s, \tau_p)$
  if fresh-engine replay of $\mathcal{D}$ from s_0 executes every action, connects all nets, and yields zero DRVs **then**
    return $(s_0, \mathcal{D})$ ▷ accept
  if the first divergent action lies in no τ_p **then**
    return FAIL
  Prune p if its τ_p holds the first divergent action for the third time
return FAIL

$$f(\tau_p, a) = 2(|\tau_p| + 1) + \delta_{\mathrm{mode}}(a), \qquad \delta_{\mathrm{mode}}(a) = \begin{cases} 0, & \text{walkaround,} \\ 0.5, & \text{push-and-shove,} \\ 1, & \text{mark-obstacles.} \end{cases} \tag{7}$$

Other candidates retain Equation (2) and use walkaround mode. Ties are resolved in insertion order. The heuristic is computed once per connection using multi-source Dijkstra on an eight-connected 0.2 mm grid. The grid uses optimistic obstacles and a layer-change cost equivalent to 30 mm. If the grid field is unavailable, the search uses a Euclidean fallback. The grid guides cost estimates, detour checks, and expansion-budget reduction. It does not determine legality.

The requested endpoint $\boldsymbol{x}_a$ in Equation (2) comprises the requested coordinates and the layer after the requested operation. For a via this is a different layer. When a candidate action is selected, the search restores its parent checkpoint if necessary and executes the operation in the engine. Duplicate heads are compared at 0.01 mm precision. Candidates require an effective valid action, no increase in evaluated geometric DRVs, and no dangling-track overshoot. The initial pass also limits detours by $\Delta\ell \leq 1.3\Delta h + 1$ mm, where $\Delta h = \max(0, \min(h_{\mathrm{before}}, 10^6) - \min(h_{\mathrm{after}}, 10^6))$. The heuristic values are evaluated at the current and next heads, with $h_{\mathrm{after}} = 0$ upon connection. A failed connection is retried without this cap.

Search stops when it finds a connection, exhausts the queue, or reaches its expansion budget. Construction permits 60 expansions and final repair permits 200. A grid-unreachable target reduces the budget to at most 12. A connecting prefix is replayed linearly from its root before it is returned. The heuristic is not established to be admissible or consistent for the implemented hop cost. The finite search does not guarantee a shortest path. Search-time geometric DRC filtering excludes connectivity-related reports and dangling tracks. The post-connection filter includes dangling tracks and excludes only missing connections, which are checked separately.

Finalization and replay. The generator removes construction routing and assembles recorded connection actions with net-selection and net-ending operations. Algorithm 3 replays this sequence on the completed unrouted instance. Replay of τ_p fails when an action fails, unconnected links do not decrease, or strict-check DRVs remain. Pruning removes pad p and its sequence τ_p, and it removes nets with fewer than two remaining pads.

Final acceptance requires plain replay on a fresh engine server without search or checkpoint restoration. Its terminal DRV count omits the intermediate filters. The exact unrouted board used for this check is retained with the action sequence. Construction geometry need not be reproduced exactly during final replay. Recorded operations specify action type, target coordinates and layers, and routing mode. Replay resolves coordinates against live candidates and permits a 0.8 mm snapping fallback for via targets.

Realized counts and calibration. The mathematical net index persists through pruning. A removed net has $\hat{P}_i = 0$, giving

$$\hat{P} = \sum_{i=1}^{\bar{N}} \hat{P}_i, \qquad \hat{N} = \sum_{i=1}^{\bar{N}} \mathbf{1}[\hat{P}_i \geq 2]. \tag{8}$$

These indices are mathematical bookkeeping. Stored net codes can be renumbered, and the saved witness does not preserve a direct original-to-final net-code mapping.

For the accepted pilot boards $\mathcal{S}_r$ at scale α_r, the pilot update is

$$\alpha_{r+1} = \alpha_r \frac{m_{\text{target}}}{m_r}, \qquad m_r = \text{median}_{s \in \mathcal{S}_r} \frac{100\hat{P}(s)}{W(s)H(s)}, \tag{9}$$

where m_r is the median realized pad density in pads/cm^2. The factor 100 converts area in mm^2 to cm^2. The D3-B ref target is $m_{\text{target}} = 3.628$ pads/cm^2. The sampled quantity multiplied by α remains net density. Each pilot attempts 3,000 boards with fresh seeds. Pilots stop when m_{target}/m_r lies within 2% of one or after four iterations. The pilots used scales 1.0000, 1.3324, 1.5191, and 1.6074. Its final pilot ratio was 1.0268, outside the tolerance. Production used the subsequent update, 1.6505, without another pilot. The procedure therefore stopped at its iteration limit without meeting the tolerance. Median correction does not establish agreement of distribution tails or dependencies between statistics.

A.2 Acceptance Rate and Runtime

We report costs for one server with two AMD EPYC 7453 CPUs (56 cores) running 80 worker processes. The times below follow from the throughput measured on each of 17 such servers, which generated boards in parallel without coordination. Generation accepts about 920 boards per hour on one server. Production attempted 106,000 boards and accepted 105,066 (99.1%), which takes about 115 hours on one server. The remaining 934 boards failed finalization or raised an error. Most rejections occur at the pad level and remove only the rejected pad, so few boards are discarded. The median wall-clock time per accepted board was 106.5 seconds, and the 90th percentile was 845 seconds. The 90th percentiles were 783 seconds for construction, 741 seconds for TryConnect search within it, and 59 seconds for replay with repair and pruning. The four calibration pilots take about 13 hours, and the independent replay of the first 100,000 accepted demonstrations takes about 18 hours. The full pipeline therefore takes about 146 hours, or six days, on one server. Every replayed demonstration executed all actions, connected all nets, and yielded zero DRVs.

B Training Details

Architecture and PPO. All policies used the Transformer policy of PCBWorld (Song et al., 2026), enlarged from four to six layers, width 128 to 256, four to eight attention heads, and feed-forward width 512 to 1,024. For a fair comparison, we retrained our reimplementation of PCBWorld, PPO (D2), at this enlarged size, and it exceeds the original policy in CP on D3-A. Table 7 lists the architecture and PPO hyperparameters shared by all runs. These runs shared training seed 42. The results therefore do not measure variation across training seeds.

The policies shared the configured reward rule, DRC observations, multi-resolution directional candidates, off-board masking, and connectivity filtering. No run overrode the reward coefficients. PPO (D2) used the same Transformer dimensions but omitted obstacle, shape, and per-net constraint features. It represented outlines with polygonal segments and encoded edges through their endpoints. The RouteDemos policies additionally encoded an edge midpoint to represent native arcs. PPO (D3-B ref) used the same architecture and full observation settings as scratch training on RouteDemos.

Behavior cloning. BC trained the same policy from random initialization by maximizing the log-likelihood of the recorded actions in the 100,000 RouteDemos training demonstrations. It used the action masks of the PPO update, and replay of each demonstration in the engine reconstructed its states. Optimization used AdamW with $(\beta_1, \beta_2) = (0.9, 0.999)$, $\epsilon = 10^{-5}$, and weight decay 10^{-4}. The learning rate warmed up linearly over 2,000 steps to 4×10^{-4} and then decayed by a cosine schedule to 5% of this peak. Training ran for 350,000 steps with 512 state–action pairs per

Table 7: Architecture and PPO hyperparameters shared by all policies. Table 8 lists the settings that differ between runs.

Hyperparameter	Value
Architecture	
Transformer layers	6
Embedding width	256
Attention heads	8
Feed-forward width	1,024
Parameters	6.79M
PPO	
Parallel environments	64
Rollout steps per environment	1,024
Update epochs	4
Batch size	128
Learning rate	2×10^{-5}
Clipping parameter	0.1
Discount factor	0.995
Generalized advantage estimation (GAE) parameter	0.95
Reward normalization	Enabled
Advantage normalization	Enabled
Entropy coefficient	0
Training seed	42

step, gradient-norm clipping at 1.0, bf16 autocast, and seed 0. Validation negative log-likelihood (NLL) was computed every 2,000 steps on 2,048 state–action pairs from 128 held-out RouteDemos validation demonstrations. Among the checkpoints saved every 35,000 steps, the one at step 280,000 had the lowest validation NLL (0.819) and initialized RL fine-tuning. No real board entered BC training or checkpoint selection.

Fine-tuning and scratch PPO. CADET fine-tuned the BC policy for 200 PPO iterations with the KL penalty of Equation (4) and $\beta = 0.05$. Scratch PPO trained from random initialization for 500 iterations without the KL penalty. BC does not train the critic, so fine-tuning began with a critic warm-up. Critic warm-up froze the actor for at least ten iterations. It stopped when critic explained variance reached 0.9 for three consecutive iterations, with a 50-iteration cap. With critic detachment, the critic loss does not update the encoder shared with the actor. Scratch runs used neither setting. Table 8 summarizes the differing settings.

Table 8: Configured policy differences. Scratch denotes PPO on RouteDemos, its variant without TryConnect, or D3-B ref. Warm-up values denote configured minimum and maximum lengths, not measured durations.

Setting	CADET	Scratch	PPO (D2)
Initialization	BC, step 280k	Random	Random
Obstacle / shape / net constraints	Enabled	Enabled	Disabled
Outline representation	Native arcs	Native arcs	Polygonal
Edge midpoint encoding	Enabled	Enabled	Disabled
KL coefficient	0.05	0	0
Critic warm-up (iterations)	10–50	0	0
Critic detachment	Enabled	Disabled	Disabled

C EVALUATION METRICS

All metrics are computed from the final board file of a rollout with the post-hoc KICAD DRC of §4.1. Let $U(s)$ be the number of ratsnest lines, i.e., unrouted connections, that KICAD reports in s. Let $n_{\mathrm{err}}(s)$ be the number of error-severity DRVs, which include unrouted connections, and let $\ell(s)$ and $n_{\mathrm{via}}(s)$ be the wirelength and via count of §2.2. For a rollout from s_0 that ends in s_T, the

per-board metrics are

$$\mathrm{CP} = \mathbf{1}\big[n_{\mathrm{err}}(s_T) = 0\big], \qquad \mathrm{Rout} = 1 - \frac{U(s_T)}{U(s_0)}, \qquad \Delta\Phi = \Phi(s_T) - \Phi(s_0), \tag{10}$$

$$\mathrm{WL} = \ell(s_T), \qquad \mathrm{Vias} = n_{\mathrm{via}}(s_T), \qquad \mathrm{DRVs} = n_{\mathrm{err}}(s_T). \tag{11}$$

A rollout that produced no board file counts as unrouted with $s_T = s_0$. A run scores a metric m by its mean $\frac{1}{|\mathcal{E}|}\sum_{b\in\mathcal{E}} m_b$ over the boards b of an evaluation set $\mathcal{E}$. Tables report the mean of this score over runs, and Table 9 adds its standard deviation.

Selection @k. In a run of a method @k, board b has k rollouts with metrics $\mathrm{CP}_b^{(j)}$ and $\Delta\Phi_b^{(j)}$ for $j = 1, \ldots, k$. The run keeps the rollout

$$j_b^\star = \operatorname*{arg\,max}_{j\in\{1,\ldots,k\}} \big(\mathrm{CP}_b^{(j)},\, \Delta\Phi_b^{(j)}\big), \tag{12}$$

where the maximum compares pairs lexicographically, and all metrics of board b except Time come from rollout $j_b^\star$. CADET ran 12 rollouts per board. CADET @1 uses the first four as four runs, and CADET @3 splits all 12 into four runs of three consecutive rollouts.

Time. Let $t_b^{(j)}$ be the wall-clock time of rollout j on board b, measured with one board at a time on the server of §4.1. Time is the board mean of $\mathrm{Time}_b = \sum_{j=1}^{k} t_b^{(j)}$, so @$k$ pays for all k rollouts. For learned policies, $t_b^{(j)}$ covers board loading, environment reset, the episode, and board saving, and excludes model loading. For LLM agents, it covers the routing episode. For rule-based routers, it covers router startup, input parsing, routing, and output conversion, and a run that reaches the 1,800-second limit counts as 1,800 seconds. Time excludes the post-hoc DRC, including the DRC used for @k selection.

D ROUTING ENVIRONMENT

This section summarizes the parts of the environment in §2.2 that the paper uses. Song et al. (2026) give the full specification.

Episode. An episode starts from the unrouted instance s_0. It ends when all required connections are complete or at the step limit. After each action, the engine updates connectivity and reports DRVs, which enter the next state and the reward. The state also holds the routing head, which records the active net, the current position and layer, and the routing phase. At each step, the environment masks actions that the current phase does not allow, such as a line operation before a route starts.

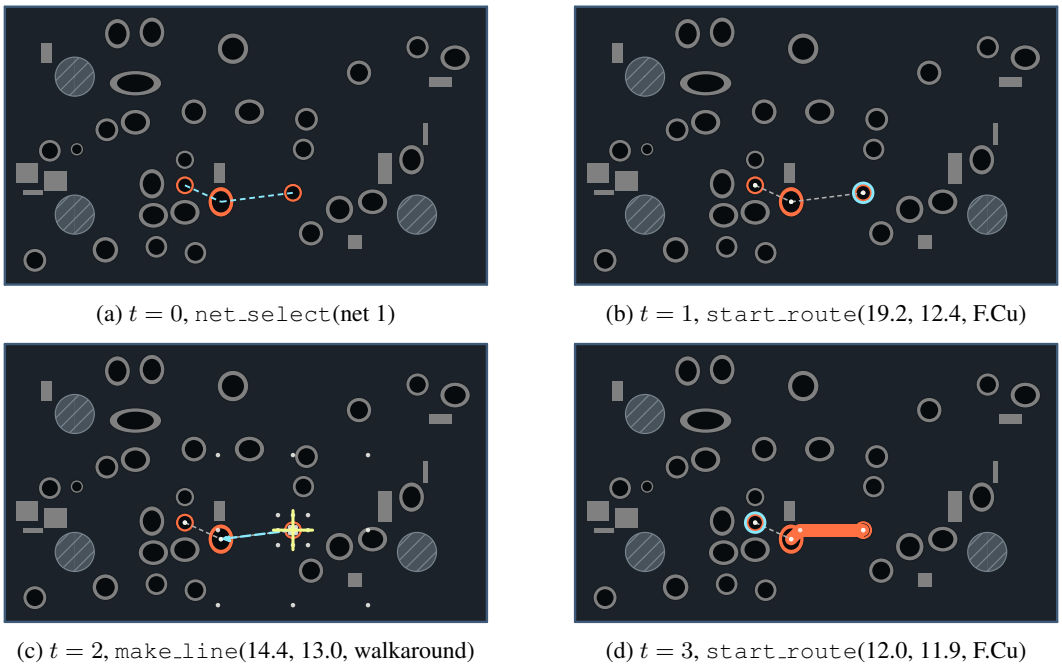
Actions. We use five engine-level actions.

- `net_select`(i) makes net i the active net and applies its rule vector $\boldsymbol{r}_i$ in the engine.
- `start_route`($\boldsymbol{x}$, l) places the routing head at location $\boldsymbol{x}$ on layer l, such as a pad or the end of existing copper of the active net.
- `make_line`($\boldsymbol{x}_a$, m) routes from the routing head toward the requested endpoint $\boldsymbol{x}_a$ in routing mode m. The engine constructs the track segments and moves the routing head to their end.
- `make_via`($\boldsymbol{x}_a$, m) routes to $\boldsymbol{x}_a$ as `make_line` does, places a via there, and moves the routing head to a different layer.
- `net_end` ends routing of the active net and returns to net selection.

The routing mode m selects how the engine treats blocking copper of other nets. Mark-obstacles mode stops at it, walkaround mode detours around it, and push-and-shove mode moves it aside. PCBWorld also provides a `finish` action that automatically routes to the nearest unconnected pad. We disable it in all experiments, so every connection comes from the five actions above.
Action parameterization. LLM agents write the action name and its numeric arguments as text (Appendix J). RL policies and TryConnect instead choose from the candidate action set $\mathcal{A}(s)$. Each candidate combines an action type, an endpoint, and a routing mode. Endpoints include pad and copper locations and directional points in eight directions at offsets of 0.2, 1, 5, and 25 mm per axis from the routing head. The environment masks endpoints outside the board outline.

E DEMONSTRATION EXAMPLE

This section shows the first 14 actions of a RouteDemos validation demonstration, which route two nets from s_0. Each net begins with `net_select` and ends with `net_end`. Between them, each hop pairs `start_route` with `make_line` or `make_via` (Appendices D and A.1). Each panel shows the state s_t and a_t in cyan. For actions with a location, white dots mark the candidates of a_t in $\mathcal{A}(s_t)$.



(a) $t = 0$, `net_select`(net 1)

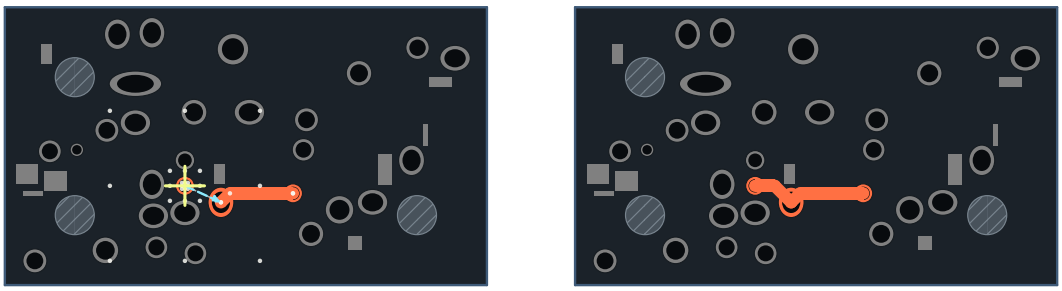
(b) $t = 1$, `start_route`(19.2, 12.4, F.Cu)

(c) $t = 2$, `make_line`(14.4, 13.0, walkaround)

(d) $t = 3$, `start_route`(12.0, 11.9, F.Cu)

Figure 4: Net 1 at $t = 0$ to 3. Each panel shows s_t and, in cyan, a_t (coordinates in mm). Orange marks the current net and gray the other nets. White dots mark candidates, and the yellow crosshair marks the routing head. A white dashed center line marks a back-layer track, and a gray dashed line an unrouted connection. In cyan, a dashed line marks `net_select`, a ring `start_route`, an arrow `make_line`, and an arrow ending in a filled dot `make_via`.

In Figure 4, `net_select` chooses net 1 at $t = 0$. Its three orange pads are to be connected, and the cyan dashed line shows their unrouted connections. At $t = 1$, `start_route` chooses a start point among the pad centers (cyan ring) and places the routing head there (yellow crosshair at $t = 2$). At $t = 2$, the white dots are the requested endpoints of `make_line` from the routing head, including the center of another pad. The chosen `make_line` ends at that pad center, and the engine routes a track to it. Each recorded hop begins with `start_route`. At $t = 3$, it chooses among the pads and track ends and places the routing head at the unconnected pad.

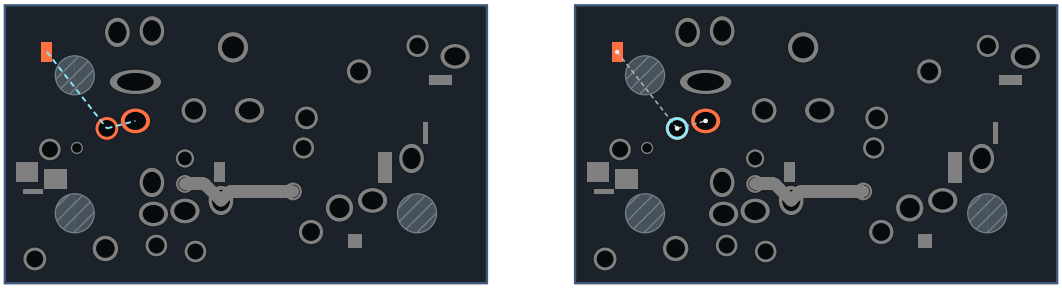


(a) $t = 4$, `make_line`(14.4, 13.0, walkaround)

(b) $t = 5$, `net_end`()

Figure 5: Net 1 at $t = 4$ and 5. Markings follow Figure 4.

In Figure 5, `make_line` connects the remaining pad at $t = 4$, and `net_end` finishes net 1 at $t = 5$.



(a) $t = 6$, `net_select`(net 2)

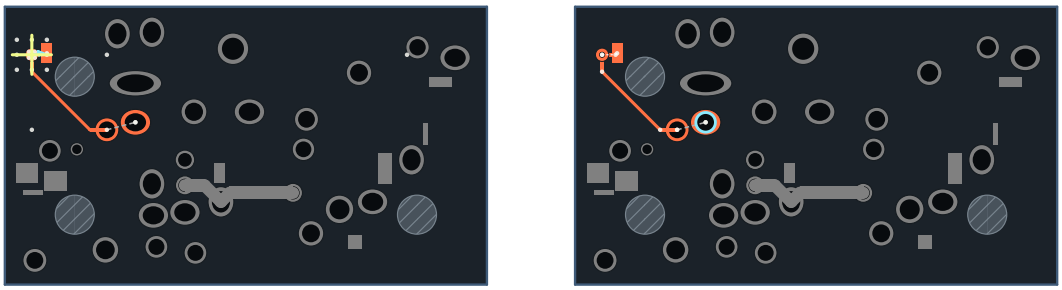
(b) $t = 7$, `start_route`(6.8, 8.2, F.Cu)

(c) $t = 8$, `make_via`(1.8, 3.2, walkaround)

(d) $t = 9$, `start_route`(1.8, 3.2, B.Cu)

Figure 6: Net 2 at $t = 6$ to 9. Markings follow Figure 4.

In Figure 6, `net_select` chooses net 2 at $t = 6$, and net 1 turns gray. At $t = 7$, `start_route` starts at the round pad. At $t = 8$, `make_via` requests a free point rather than a pad center (filled cyan dot). The engine routes a front-layer track there and places a via. This single action creates three track segments. At $t = 9$, `start_route` starts from the via on the back layer.



(a) $t = 10$, `make_line`(2.8, 3.1, walkaround)

(b) $t = 11$, `start_route`(8.7, 7.7, F.Cu)

(c) $t = 12$, `make_line`(6.8, 8.2, walkaround)

(d) $t = 13$, `net_end`()

Figure 7: Net 2 at $t = 10$ to 13. Markings follow Figure 4.

In Figure 7, a short back-layer `make_line` reaches the surface-mount pad on that layer at $t = 10$. At $t = 11$, `start_route` chooses the remaining unconnected pad, and `make_line` connects it to the round pad at $t = 12$. At $t = 13$, `net_end` finishes net 2.

F Reward Potential

This section specifies the potential Φ of §2.2. Let $\mathcal{R}$ be the set of nets with at least two pads, and let $U(s)$ be as in Appendix C. For net i in state s, let $g_i(s)$ be the number of connected components formed by its pads and copper. Let $N_{\mathrm{con}}(s) = |\{i \in \mathcal{R} : g_i(s) = 1\}|$ be the number of fully connected nets. Unlike n_{err}, the DRVs in Φ also include dangling tracks, dangling vias, and net conflicts. Let $n_{\mathrm{drv}}(s)$ be their total count, $\mathcal{V}(s)$ the set of nets with at least one of them, and $v_k(s)$ their count on net k. The set $\mathcal{V}^+(s)$ adds to $\mathcal{V}(s)$ one entry $k = 0$ that collects DRVs attributed to no net, when such DRVs exist. The potential is

$$\Phi_{\mathrm{con}}(s) = -U(s) + N_{\mathrm{con}}(s) + |\mathcal{R}| - |\mathcal{V}(s)| + b_{\mathrm{done}}\big(\mathbf{1}[U(s) = 0] + \mathbf{1}[n_{\mathrm{drv}}(s) = 0]\big), \tag{13}$$

$$f_{\mathrm{drv}}(s) = 3\ln\Big(1 + \frac{|\mathcal{V}^+(s)|}{2}\Big) + \sum_{k \in \mathcal{V}^+(s)} \ln\Big(1 + \frac{v_k(s)}{2}\Big), \tag{14}$$

where $b_{\mathrm{done}} = \frac{1}{2}\sum_{i \in \mathcal{R}} g_i(s_0)$ is computed at each reset. The quality weights are $\lambda_w = 0.5/L$ with $L = \max(W, H)$ and ℓ in millimeters, and $\lambda_v = 0.2$. The reward is

$$r_t = \Phi(s_{t+1}) - \Phi(s_t) - 0.01\,\mathbf{1}[a_t \text{ is invalid}], \tag{15}$$

where an action is invalid if the engine rejects it or it has no effect. No terminal reward is added. The penalty enters only r_t, so $\Delta\Phi$ in Appendix C excludes it.

G Full Evaluation Results

Table 9 reports all evaluated methods with standard deviations over runs and three additional layout metrics, WL, Vias, and DRVs of Equation (11). Unrouted connections contribute no tracks, so lower routability tends to shorten wirelength.

Table 9: Full evaluation results as mean ± standard deviation over four runs. KRT and the LLM agents ran once and have no standard deviation. All methods except CADET @3 use one rollout per run. CP and Rout. are in percent, WL is in millimeters, and Time is in seconds per board.

Method	CP	ΔΦ	Rout.	WL	Vias	DRVs	Time
D3-B test (141)							
Reference	100.0	144.41	100.0	575	6.6	0.00	–
FreeRouting @1	65.6±4.1	119.33±3.22	96.8±0.1	527±5	4.1±0.2	1.14±0.22	45.6±6.0
KRT @1	27.7	84.71	78.3	463	14.7	8.23	2.7
GPT-6 Astra @1	56.0	104.20	86.0	466	7.5	6.81	491.7
GPT-6 Luna @1	5.0	37.72	47.9	203	2.0	23.37	335.7
PPO (D2)	37.4±3.4	78.81±2.47	71.9±1.2	597±10	11.4±0.6	13.32±0.47	–
PPO (D3-B ref)	33.2±3.0	91.31±2.39	91.1±0.8	757±16	3.6±0.2	4.11±0.37	–
PPO (RouteDemos)	38.7±0.4	95.86±0.21	92.2±0.3	700±4	4.9±0.2	3.69±0.13	–
PPO (RouteDemos w/o TryConnect)	34.0±3.1	93.15±1.32	91.9±0.5	702±11	4.0±0.1	3.78±0.25	–
CADET (BC)	49.3±2.4	82.00±2.41	83.8±2.0	777±25	13.4±0.6	7.38±0.80	–
CADET @1	66.7±1.5	107.60±1.65	90.7±1.0	654±19	9.7±0.3	4.58±0.53	14.1±0.3
CADET @3	80.3±2.0	120.96±1.20	95.4±0.2	647±8	8.8±0.4	2.32±0.18	43.4±1.6
D3-A (99)							
Reference	100.0	50.61	100.0	170	1.2	0.00	–
FreeRouting @1	82.1±1.9	46.05±0.52	97.7±0.3	153±1	1.0±0.0	0.38±0.05	8.6±1.6
KRT @1	69.7	40.54	92.8	146	3.8	1.42	1.5
GPT-6 Astra @1	83.8	45.03	93.8	138	1.8	0.97	138.9
GPT-6 Luna @1	46.5	31.30	78.5	119	1.2	2.90	179.1
PPO (D2)	75.3±1.0	40.93±0.76	87.0±1.4	176±4	2.1±0.4	1.88±0.19	–
PPO (D3-B ref)	74.0±3.8	42.20±1.05	95.2±0.5	202±3	0.9±0.0	0.65±0.05	–
PPO (RouteDemos)	79.5±1.7	44.77±0.71	96.9±0.4	192±2	1.4±0.1	0.39±0.04	–
PPO (RouteDemos w/o TryConnect)	76.3±1.0	43.72±0.35	96.7±0.2	191±2	1.1±0.0	0.44±0.02	–
CADET (BC)	83.3±2.4	41.81±0.76	94.4±0.6	196±7	3.1±0.2	0.85±0.11	–
CADET @1	91.4±1.0	45.17±0.67	97.5±0.3	185±4	2.2±0.2	0.36±0.10	3.8±0.2
CADET @3	97.5±1.0	48.65±0.49	99.3±0.3	173±2	1.7±0.1	0.09±0.03	11.6±0.2

H TRAINING CURVES

Figure 8 shows D3-B test performance during training. RL checkpoints were evaluated every ten iterations on all 141 D3-B test boards. Each value averages five rollouts per board and then averages over boards, so it differs from the @1 protocol of §4.1. D3-B test did not influence checkpoint selection. RL curves use a centered moving average over five evaluation points. The CADET (BC) curve shows the BC checkpoints saved every 35k steps up to the 280k checkpoint that initializes fine-tuning. Its horizontal axis maps 280k BC steps to 300 iterations, and its initial value is the mean random-initialization value of the scratch runs. CP rose to 48.1% during BC and to 66.4% after 200 fine-tuning iterations. No scratch PPO run exceeded 40% CP within 500 iterations.

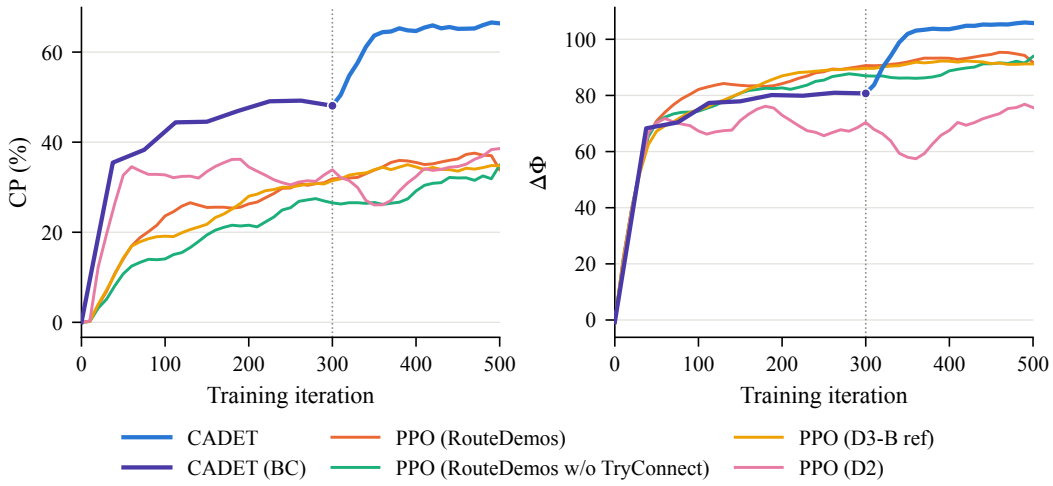


Figure 8: D3-B test CP and $\Delta\Phi$ during training, averaged over five rollouts per board. CADET (BC) steps are rescaled so that 280k steps span 300 iterations. The dotted line marks the start of CADET fine-tuning.

I REPRODUCIBILITY DETAILS

Table 10 lists where the paper specifies each component needed to reproduce RouteDemos and the reported results. Anonymized source code for generation, training, and evaluation, together with the D3-B split definitions, is available at `https://anonymous.4open.science/r/CADET-FBBF`. We release RouteDemos and the trained checkpoints.

Table 10: Location of the specification for each component.

Component	Specification
Board attribute distributions and fitted values	§3.1, Appendix A.1, Table 6
Construction and TryConnect search	Algorithms 1 and 2, Appendix A.1
Replay, repair, pruning, and acceptance	§3.2, Algorithm 3
Pad-count calibration	Appendix A.1, Equation (9)
Generation cost and acceptance rate	Appendix A.2
Data splits and board counts	§4.1, Appendix A
Policy architecture and PPO settings	Appendix B, Tables 7 and 8
BC settings and checkpoint selection	Appendix B
Evaluation protocol and metrics	§4.1, Appendix C
Reward potential	§2.2, Appendix F
Software, compute, and rule-based baselines	Appendix I
LLM agent settings and prompts	Appendix J

Software. BC, PPO training, and evaluation all used KiCad 9.0.8, Python 3.12, PyTorch 2.8.0 with CUDA 12.8, and Gymnasium 1.2.3. The rule-based baselines were FreeRouting v2.1.0 and KRT v0.15.1.

Compute. BC took about 97 hours on four NVIDIA RTX PRO 6000 GPUs. Each PPO run used one NVIDIA L40 or A10 GPU and 64 environments. CADET fine-tuning took about 85 hours, and scratch PPO took 20 to 177 hours, depending on the training set. Checkpoint evaluation on D3-B ref ran separately on NVIDIA A10 GPUs.

Baseline settings. FreeRouting and KRT had a time limit of 1,800 seconds per run. KRT used a 0.1 mm grid, at most three rip-ups, and heuristic weight 1.9, with layers, track widths, clearances, and via sizes read from each board.

J LLM Agent Setup

Both LLM agents used medium reasoning effort, at most 4,000 output tokens per step, and the last four steps as history. An LLM episode ended after ten consecutive steps without geometric progress. The API cost of one run was $1,131.51 on D3-B test and $111.21 on D3-A for GPT-6 Astra, and $6.10 and $1.54 for GPT-6 Luna.

Each LLM agent step sent a system prompt and a user prompt. The system prompt (Figure 9) holds the task, the routing guidelines, the response format, the S-expression state format, and the static board context. The user prompt (Figure 10) holds the current step, the dynamic board state, the action history, and the valid actions. Table 11 describes the placeholders. The agent parsed its action from the `<action>` block of each response.

Table 11: Placeholders in the LLM agent prompt.

Placeholder	Content
{state_format_desc}	State format description in Figure 10 (top), inserted into the system prompt
{board_static}	Board outline, nets with pad locations, and obstacles in S-expression form
{current_step}	Current step index
{dynamic_observation}	Per-net tracks, vias, and unconnected points, and the routing head state
{valid_step_count}	Number of valid actions taken so far
{action_history}	Last four steps, each marked "[no effect]" if it made no progress
{rejected_attempts}	Empty unless the last action was refused, otherwise a "Last rejected attempt" block
{valid_actions}	Actions allowed in the current routing phase

```
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 ≥1mm from any
    pad — never place a via on a pad), route around on the opposite layer, then switch
    back.
- If a detour looks too long or also blocked, use PushAndShove mode (`p`) on `
   make_line` / `make_via` to shove other nets' tracks aside and push the current
   route straight through.
- When the cursor is already near the remaining target with a clear path, draw the
   final segment directly with `make_line <x> <y> <mode>` onto the target pad's exact
   (x, y).
- When the current net's (points ...) list is empty, the net is done — issue `net_end`
    explicitly to release it before moving on. Continuing with `make_line` / `make_via
   ` on a completed net wastes steps and may trip the no-progress streak.

## 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 the final `make_line` cannot land
   on it.
Always make the layer-return `make_via` at an offset waypoint (≥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
{state_format_desc}

{board_static}
```

Figure 9: Full system prompt of the LLM agents.

```
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
       (MarkObstacles: stop at blockers; Walkaround: detour around them; PushAndShove:
    shove other nets' blocking tracks aside to force the current route through)
  (step N ratio)        — current step number and step/max_steps ratio
```

```
## 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}
```

Figure 10: State format description inserted into the system prompt (top) and user prompt template (bottom) of the LLM agents.

K DATASHEET FOR ROUTEDEMOS

This appendix summarizes RouteDemos following the main categories of Datasheets for Datasets (Gebru et al., 2021).

Motivation. RouteDemos provides engine-level demonstrations for PCB routing. Public board collections hold completed boards, not demonstrations. We used RouteDemos to train routing policies by BC (§3.3).

Composition. RouteDemos contains 100,000 training and 128 validation instances. Each instance is a synthetic two-layer board. It includes the unrouted board and its design rules in KICAD format, its demonstration, the routed board that the demonstration produces, and the seed and sampled board attributes. A demonstration is a sequence of engine-level actions, each stored as its type, coordinates, layer, and routing mode. The dataset contains no personal information.

Collection process. CADET generated all instances with D3-B ref as $\mathcal{D}_{\mathrm{ref}}$ (§3.2). Real boards enter only through the attribute distributions and the pad-count calibration. No real-board geometry is copied. Each board is accepted only when fresh-engine replay of its demonstration connects all nets with zero DRVs. Appendix A.2 reports the acceptance rate and runtime.

Uses. RouteDemos is intended for learning PCB routing policies from demonstrations in the engine-level action space. Its attribute distributions are estimated from the two-layer boards of D3-B ref. Each demonstration is one valid routing, not an optimal one.

Distribution and maintenance. RouteDemos and its generator are released under the BSD-3-Clause license, as are the synthetic datasets of PCBWorld. The D3-B boards come from PCBench (He et al., 2024) and keep the licenses of their source repositories. The release therefore excludes these boards and includes only the statistics that the generator reads. Each release is versioned with the generator and the KICAD version used for validation.