Cypress: VLSI-Inspired PCB Placement with GPU Acceleration

Niansong Zhang*
nz264@cornell.edu
Cornell University
Ithaca, NY, USA

Anthony Agnesina
aagnesina@nvidia.com
NVIDIA Corporation
Atlanta, GA, USA

Noor Shbat
nshbat@nvidia.com
NVIDIA Corporation
Yokneam, Israel

Yuval Leader
leader@nvidia.com
NVIDIA Corporation
Yokneam, Israel

Zhiru Zhang
zhiruz@cornell.edu
Cornell University
Ithaca, NY, USA

Haoxing Ren
haoxingr@nvidia.com
NVIDIA Corporation
Austin, TX, USA

Abstract

The scale of printed circuit board (PCB) designs has increased significantly, with modern commercial designs featuring more than 10,000 components. However, the placement process heavily relies on manual efforts that take weeks to complete, highlighting the need for automated PCB placement methods. The challenges of PCB placement arise from its flexible design space and limited routing resources. Existing automated PCB placement tools have achieved limited success in quality and scalability. In contrast, very large-scale integration (VLSI) placement methods have proven to be scalable for designs with millions of cells and delivering high-quality results. Therefore, we propose Cypress, a scalable, GPU-accelerated PCB placement method inspired by VLSI. It incorporates tailored cost functions, constraint handling, and optimized techniques adapted for PCB layouts. In addition, there is an increasing demand for realistic and open-source benchmarks to (1) enable meaningful comparisons between tools and (2) establish performance baselines to track progress in PCB placement technology. To address this gap, we present a PCB benchmark suite synthesized from real commercial designs. We evaluate our method against state-of-the-art commercial and academic PCB placement tools with the benchmark suite. Our approach demonstrates a 1–5.9× higher routability on the proposed benchmarks. For fully routed designs, Cypress achieves 1–19.7× shorter routed track lengths. With GPU acceleration, Cypress delivers up to 492.3× speedup in run time. Finally, we demonstrate scalability to real commercial designs, a capability unmatched by existing tools.

CCS Concepts

• **Hardware → PCB design and layout**; **Placement**; **Design databases for EDA**.

*This work was conducted during an internship at NVIDIA.

ISPD '25, Austin, TX, USA.

ACM ISBN 979-8-4007-1293-7/25/03
https://doi.org/10.1145/3698364.3705346

Keywords

Printed Circuit Board; Placement; Routing; GPU Acceleration

ACM Reference Format:
Niansong Zhang, Anthony Agnesina, Noor Shbat, Yuval Leader, Zhiru Zhang, and Haoxing Ren. 2025. Cypress: VLSI-Inspired PCB Placement with GPU Acceleration. In *Proceedings of the 2025 International Symposium on Physical Design (ISPD '25), March 16–19, 2025, Austin, TX, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3698364.3705346

1 Introduction

PCB placement and routing (PnR) is typically carried out manually by PCB layout engineers. However, with the increasing integration density and scale of PCB design, this manual process takes many weeks. Approximately 50% of the total PCB design time is spent on component placement [37], making it a significant bottleneck in the design process. Despite decades of research on design automation, the industry has yet to adopt automated PCB placement tools widely. An important reason is that existing PCB placement tools struggle with large-scale designs and achieve limited placement quality. For example, the state-of-the-art commercial PCB PnR solution Quilter [37] takes 2.6 hours to layout a design with 176 components. RePlace [8] fails to find a routable placement for multilayer PCB designs with components of diverse sizes [7].

The challenge of PCB PnR lies in its complex design space and limited routing resources [30]. Unlike VLSI, PCB placement has a flexible design space with diverse-sized components that can be rotated and placed on both sides of the board. The shape of the PCB board and the components are often irregular. VLSI routing utilizes rectilinear single nets with many small vias across numerous routing layers, while PCBs employ bus-level traces that allow greater freedom in angles but typically use fewer routing layers.

In contrast, VLSI placement methods are widely used in standard cell IC designs and mixed-size system-on-chip (SoC) designs. Analytical methods such as mPL6 [3], NTUplace [4, 5, 20], ePlace [31–33] formulate placement as a mathematical optimization problem. DREAMPlace-based methods [14, 25, 27, 28] leverage gradient descent, modern machine learning optimizers, and GPU acceleration to accelerate placement and improve solution quality. These VLSI placement algorithms have been proven to be scalable for designs with millions of cells and deliver high-quality results. Therefore, we propose Cypress [1], a VLSI-inspired PCB placement approach with tailored cost functions, constraint handling, and optimization

[1]Cypress and the PCB placement benchmark suite is open-source and available at https://github.com/NVlabs/Cypress.

techniques to improve both the design quality and the scalability. Although VLSI-inspired methods enhance scalability and placement quality, there is no single solution for PCB placement. Critical components, such as those for high-current or high-speed signals, often require careful manual placement and routing. Our work focuses on a fast placement routine for noncritical components, helping designers accelerate the iterative process needed to meet performance requirements and design constraints.

We also highlight the lack of high-quality benchmarks for the PCB placement problem. To enable meaningful comparisons between tools and establish a performance baseline, we introduce a realistic open-source PCB placement benchmark consisting of 10 synthesized designs with noncritical components. These benchmarks are derived from commercial PCB designs and insight into how engineers approach manual PCB layout. Finally, we evaluate Cypress against state-of-the-art PCB placement tools from industry, open source projects, and academia, using both proposed benchmarks and large-scale commercial PCB designs.

We summarize our contributions as follows:

- **Tailored cost functions for PCB placement**. Our method introduces a novel problem formulation tailored specifically for PCB placement, incorporating orientation-aware wirelength, density, and net crossing. This formulation addresses the unique challenges of PCB design, enabling more accurate evaluation and optimization in real-world scenarios.
- **Scalability**. Cypress efficiently handles large-scale PCB designs with GPU acceleration. Evaluation results show up to 492.3× run time improvement over the current state-of-the-art solutions.
- **Open source benchmark**. We propose a synthesized PCB placement benchmark suite based on real PCB designs as well as related tools to visualize and interface with commercial and open-source EDA tools.
- **Evaluation on real commercial designs**. Cypress is evaluated on industrial PCB designs with thousands of components. These tests confirm the practicality and effectiveness of our methods across realistic sizes and complexities.

2 Related Work

The automation of PCB placement intersects several disciplines, including electronic design automation (EDA), optimization algorithms, and machine learning. This section reviews related literature that has contributed to the development of techniques in these areas, with a particular focus on PCB and VLSI placement.

2.0.1 Adapted VLSI Placement Algorithms. VLSI placement research has produced numerous algorithms to optimize component layouts to minimize wirelength, delay, and other metrics. Techniques such as simulated annealing [13, 16, 34], genetic algorithms [17, 23, 42], particle swarm optimization [6, 40], and more recently, deep learning approaches [2, 41], have been adapted to address the PCB placement problem. However, their practical use is hindered by scalability challenges and a failure to account for PCB designs' unique flexibility, such as component rotation, dual-sided placement, and the need to accommodate larger, more irregular components. Directly applying VLSI placement algorithms to the PCB domain proves ineffective: the standard RePlace algorithm [8] struggles to produce suitable solutions for multi-layer designs with components of diverse sizes and shapes [7].

2.0.2 PCB Placement Specialization. Several works have introduced PCB-specific cost functions and methods for handling constraints. For instance, NS-Place [7] addresses PCB routability by formulating the problem as separating net convex hulls, a formulation closely related to support vector machines (SVM). Additionally, some efforts have focused on managing component rotation. Hsu et al. [18] extend the log-sum-exp differentiable wirelength model to account for arbitrary component rotation, while Lin et al. [26] refine this approach by defining a mapping function that translates arbitrary rotations into valid orientations.

2.0.3 Machine Learning in PCB Placement. The application of machine learning to PCB placement is relatively recent. Several approaches have attempted to use neural networks and reinforcement learning to predict more optimal placements [15]. These methods aim to capture the tacit knowledge of experienced designers and translate it into actionable placement strategies. Notably, some frameworks introduce learnable multi-objective cost models that adapt to the specificities of each PCB project, offering a promising avenue for managing the complex trade-offs involved in placement decisions [1].

2.0.4 Commercial and Open Source PCB Placement Tools. Several automated PCB placement tools exist in both commercial and open-source domains. Cadence Allegro X AI employs a decision-making approach to PCB placement, utilizing Monte Carlo Tree Search (MCTS) to solve placement challenges [21, 24]. However, this method is typically constrained to smaller designs due to substantial computational and memory demands. Quilter [37] uses reinforcement learning to explore the design space, performing placement and routing simultaneously. Its current capabilities support boards with fewer than 1,000 pins, 100 components, 10% density, 500 MHz signals, and 2 A currents. Moreover, generating design candidates can take up to several hours. In the open-source community, the OpenROAD Project offers a simulated annealing-based PCB placer (SA-PCB) that supports component rotation [10].

3 Preliminaries

In this section, we introduce key concepts and workflows relevant to modern PCB design and the state-of-the-art placement algorithm.

3.1 Commercial PCB Design

In the modern commercial PCB design process, the workflow begins with schematic capture, where engineers create a circuit diagram that defines the electrical connections between components. Then follows part selection, where suitable components are chosen based on performance, cost, and availability. The design is then translated into a netlist, a data structure that lists all the connections between components. After the schematic is complete, the physical design phase begins. Here, components are placed on the PCB layout, and their positions must satisfy a variety of constraints, including electrical performance, signal integrity, thermal management, and mechanical considerations. PCB components can be rotated during placement. However, they cannot be rotated to arbitrary angles;

Table 1: Notations

Notation	Description	Notation	Description
E	Set of nets	$\mathbf{x}, \mathbf{y}$	Components center locations
$\boldsymbol{\theta}$	Components orientations	$\Theta = \{o_1, o_2, ..., o_n\}$	Set of legal orientations (ordered)
c	Component c	(x_c, y_c, θ_c)	Center location/orientation of c
v_k	Pin k	$(x_k^{\text{off}}, y_k^{\text{off}})$	Offsets of v_k from (x_c, y_c)
WL, NC	Wirelength, Net Crossing	D	Density
λ_D	Density weight	λ_{NC}	Net crossing weight
$\mathcal{P}_{\mathbf{x},\mathbf{y}}$	Preconditioner for $\mathbf{x}, \mathbf{y}$	$\mathcal{P}_{\boldsymbol{\pi}}$	Preconditioner for orientation

instead, they can only be rotated to a set of legal orientations, such as 90, 180, or 270 degrees.

Once the initial layout is completed, the design undergoes routing, where traces are drawn to connect the components according to the netlist. This is followed by Design Rule Checks (DRCs) to ensure that the layout adheres to manufacturing requirements and standards. Throughout the process, the design must go through multiple rounds of peer review and verification to address potential issues related to signal integrity, power delivery, and thermal performance. A key bottleneck in this flow is the placement phase, which can consume around 50% of the overall design time [37]. Optimizing component placement is critical as it directly impacts the ease and quality of routing, subsequently the overall design performance, and manufacturability of the PCB.

3.2 DREAMPlace

DREAMPlace builds upon the ePlace [33] and RePlace [8] algorithms by leveraging GPU acceleration to achieve high-quality global placement results. DREAMPlace frames the global placement task as a wirelength minimization problem with density penalties. This problem is solved by classical optimization methods, such as gradient descent, applied to a nonlinear, unconstrained formulation:

$$\min_{\mathbf{x},\mathbf{y}} \sum_{e \in E} w_e \, \mathrm{WL}(e; \mathbf{x}, \mathbf{y}) + \lambda \, \mathrm{D}(\mathbf{x}, \mathbf{y}), \tag{1}$$

where E represents the set of nets, while $(\mathbf{x}, \mathbf{y})$ denote the cell coordinates. The term $\mathrm{WL}(e; \mathbf{x}, \mathbf{y})$ is a smoothed approximation of the half-perimeter wirelength (HPWL) for each net e, and $\mathrm{D}(\mathbf{x}, \mathbf{y})$ is a smoothed density function, modeled as the potential energy in an electrostatic system where cells act as charges. The density is computed by solving Poisson's equation through spectral methods, specifically using a two-dimensional fast Fourier transform (FFT). Net weights w_e can account for timing constraints. Additionally, a Lagrange multiplier λ is gradually increased to prevent cell overlaps. DREAMPlace calculates wirelength and density gradients through GPU-accelerated algorithms, utilizing the PyTorch framework to achieve efficient computation.

4 Cypress Framework

Our framework models PCB placement as a multi-objective optimization problem, solved through a gradient descent approach. To optimize for routability, we introduce PCB-specific cost functions tailored to the problem's unique constraints. In addition, the framework employs Bayesian optimization to effectively explore the parameter space, leading to improved solution quality. The notation is listed in Table 1.

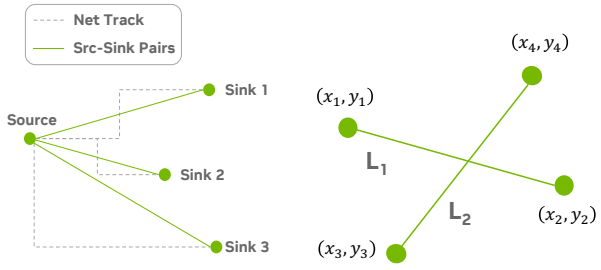
4.1 Problem Formulation

We focus on placing surface-mounted, noncritical components on a rectilinear PCB's top and bottom layers/sides. The framework takes a PCB netlist with pre-assigned layers as input. Some components, such as mechanical or high-speed, high-current critical elements, are fixed in place, and pre-routed high-speed traces can also be included as placement blockages if needed. Components can be of any shape and can be placed anywhere on the assigned layer as long as they meet spacing constraints. The goal is to assign each component c a coordinate (x_c, y_c) and a legal orientation θ_c to maximize routability. While routability is the prominent goal of PCB placement, we will realize it through optimizing the multiple proxies presented later (i.e., wirelength, net crossing, and density).

4.2 PCB-Specific Cost Functions

PCB placement differs from mixed-size VLSI placement in several key ways. First, PCB components can be placed anywhere on a layer, unlike standard cell placement in VLSI, which is constrained by fixed rows and sites. Second, PCBs have far fewer routing resources. While VLSI benefits from multiple stacked metal layers, often making routing a 3D problem, PCB routing is restricted to a much more limited 2D space, making it significantly harder to route crossing nets. Finally, PCB components can rotate to any legal orientation, a flexibility not typically considered in VLSI placement. To this end, we propose the following PCB-specific cost functions to maximize routability.

4.2.1 Net Crossing. To address the challenge of limited routing resources in PCBs, we define a metric called net crossing to capture routing conflicts on the same copper layer.



(a) Net as pin pairs (b) Pin pair crossing

Figure 1: Net crossing definition — First, we represent a multi-pin net as a set of line segments. Each line segment starts from the source pin and ends at the sink pin. The net crossing is then defined as the sum of all pin pairs' line segment crossing.

Fig. 1 illustrates the definition of net crossing. We first decompose each multi-pin net into pin pairs. As described in Algorithm 1, this transformation converts a multi-pin net into a list of pin pairs. Each pair connects the source pin to one of the sink pins with a line segment, as shown in Fig. 1(a). At the placement stage, a net is defined as the electrical connection between a source pin and its sink pins, without any information about the net's topology. The source-sink pin pair model serves as a general representation of

Algorithm 1: Transform Multi-Pin Nets to Pin-Pair Model

Input: $Nets$ // A list of multi-pin nets. Each net $N_i = (source_i, Sinks_i)$
Output: $PinPairs$ // A list of pin pairs representing the nets

$PinPairs \leftarrow \emptyset$ // Initialize the output list
foreach $N_i \in Nets$ **do**
  $source \leftarrow N_i.source$ // Extract the source pin
  foreach $sink \in N_i.Sinks$ **do**
    // Add the pin pair to the list
    $PinPairs \leftarrow PinPairs \cup \{(source, sink)\}$
  end
end
return $PinPairs$

both manual and automated routing approaches. Then, a crossing score is defined between two line segments on the same layer. As an example, we define the crossing for a pair of line segments L_1 and L_2 as depicted in Fig. 1(b). First, we express L_1 and L_2 in the first degree Bézier form:

$$L_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + t \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \end{bmatrix}, \quad L_2 = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} + u \begin{bmatrix} x_4 - x_3 \\ y_4 - y_3 \end{bmatrix} \tag{2}$$

where (x_i, y_i) is the coordinate of pin v_i, t and u are Bézier parameters defined as follows:

$$\begin{aligned} t &= \frac{(x_1 - x_3)(y_3 - y_4) - (y_1 - y_3)(x_3 - x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4) + \epsilon} \\ u &= -\frac{(x_1 - x_2)(y_1 - y_3) - (y_1 - y_2)(x_1 - x_3)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4) + \epsilon} \end{aligned} \tag{3}$$

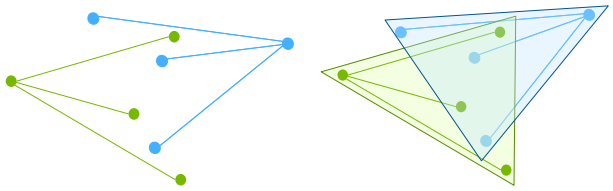
where we introduce a small value ϵ to prevent division by zero, such as when L_1 and L_2 are parallel. There is an intersection of L_1 and L_2 if $0 \le t \le 1$ and $0 \le u \le 1$ simultaneously. We apply a bell-shaped function similar to the one in the density function of NTUPlace3 [5] to obtain a smooth cost function:

$$\text{Crossing}(L_1, L_2) = B(t - 0.5)B(u - 0.5) \tag{4}$$

where B is the bell-shaped function defined as follows:

$$B(x) = \begin{cases} 1 - \nu x^2, & 0 \le |x| \le 0.5 \\ \mu\,(|x| - \sigma)^2, & 0.5 \le |x| \le 1 \\ 0, & |x| \ge 1. \end{cases} \tag{5}$$

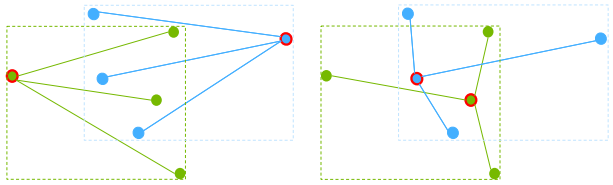
ν, μ, σ are tunable parameters subject to $1 - 0.25\nu = \mu(0.5 - \sigma)^2$. The cost in (4) will be strictly positive when segments cross, serving as penalty. Finally, the net crossing NC for the entire design is the sum of all crossings between pairs of line segments on the same layer, i.e., $\text{NC} = \sum_{(i,j)} \text{Crossing}(L_i, L_j)$ where L_i and L_j are on the same layer. The layer of two line segments can be determined based on their pins: if all four pins of the segments belong to the same layer, the segments are considered to be on the same layer. Note that crossings are only calculated for line segments on the same layer, as segments on different layers do not physically intersect and therefore do not pose a conflict in routing resources. In practice, this can be computed very fast in parallel on GPUs despite the quadratic complexity.



(a) Net crossing (b) Net convex hull

Figure 2: The net crossing provides a more accurate routing model than the convex hull. The two nets (green and blue) do not have a routing resource conflict in our net crossing definition. However, the net convex hull model inaccurately identifies a conflict between them.

NS-Place [7] developed a concept similar to net crossing by separating the convex hulls of nets, resulting in a problem formulation that closely resembles a support vector machine (SVM). We argue that the net convex hull model is too restrictive — no overlap between convex hulls implies zero net crossings — and less accurate than our net crossing model. As shown in Fig. 2, the two nets are represented as pin pairs in Fig. 2(a) and convex hulls in Fig. 2(b). When calculating net crossings in Fig. 2(a), the result is 0, which is accurate because the two nets do not have a routing resource conflict. However, in the convex hull model, overlap between convex hulls indicates a routing conflict, which is inaccurate. The pin pair-based approach better reflects how routers handle nets in PCB designs.



(a) No routing conflict (b) Routing conflict present

Figure 3: Two net pairs with different source pins (marked in red) but with the same congestion score from RUDY. (a) shows no routing conflict, while (b) presents a routing conflict.

In VLSI placement, the RUDY model [38] is widely used to estimate routing congestion, a typical proxy for routability. RUDY is calculated based on the available routing resources in a routing g-cell and its overlap with the net's bounding box. This model works well for VLSI routing because VLSI designs have grid-based rectilinear routing systems with many metal layers, and routers generally aim to route each net within the bounding box enclosing all its pins. Multi-pin nets are also routed with a unique tree, and not independent source-sink connections. Thus, RUDY is less effective for PCBs, which have fewer metal layers, and where the specific connections between pin pairs play a much larger role and can be realized with more freedom (i.e., no grid and possible

angled routing). Fig. 3 illustrates this: two cases with identical routing congestion RUDY scores show one with a routing conflict and one without, highlighting that RUDY is a flawed model for PCB routability.

4.2.2 Orientation-aware Wirelength Model. Given PCB components' large and diverse sizes, their orientations can significantly impact placement quality, especially in terms of wirelength. We extend the differentiable log-sum-exp (LSE) wirelength model [36] to consider legal orientations. $\mathrm{WL}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})$ is the total wirelength for all nets with components rotations $\boldsymbol{\theta}$ considered:

$$\begin{aligned}\mathrm{WL}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = \quad & \sum_{e \in E} \gamma \left(\log \sum_{v_k \in e} \exp\left(\frac{x_k}{\gamma}\right) + \log \sum_{v_k \in e} \exp\left(-\frac{x_k}{\gamma}\right) \right. \\ & \left. + \log \sum_{v_k \in e} \exp\left(\frac{y_k}{\gamma}\right) + \log \sum_{v_k \in e} \exp\left(-\frac{y_k}{\gamma}\right) \right),\end{aligned}$$

where (x_k, y_k) is the coordinate of pin v_k:

$$\begin{aligned} x_k &= x_c + x_k^{\text{off}} \cos\theta_c - y_k^{\text{off}} \sin\theta_c, \\ y_k &= y_c + x_k^{\text{off}} \sin\theta_c - y_k^{\text{off}} \cos\theta_c, \end{aligned} \tag{6}$$

subject to $\theta_c \in \Theta$. Parameter γ is a smoothness parameter that controls the HPWL approximation accuracy (as $\gamma \to 0$). We relax the categorical choice of a legal orientation to a softmax over all choices using the Gumbel-Softmax reparameterization technique [19]. This produces an n-dimensional sample vector $\boldsymbol{\kappa}$, where $n = |\Theta|$, and each element is defined as:

$$\kappa_i = \frac{\exp\left(\left(\log\left(\pi_i\right) + g_i\right)/\tau\right)}{\sum_{j=1}^{n} \exp\left(\left(\log\left(\pi_j\right) + g_j\right)/\tau\right)} \quad \text{for } i = 1, \ldots, n. \tag{7}$$

In Eq. 7, g_i are i.i.d samples drawn from $\mathrm{Gumbel}(0, 1)$, and τ is the softmax temperature. With this approach, the task of choosing a legal orientation is relaxed to learning the continuous class probability π [19]. During the orientation optimization, we use the continuous approximation of θ_c, defined as $\hat{\theta}_c = \sum_{i=0}^{n} o_i \cdot \kappa_i$. After the orientation optimization, we determine the legal orientation choice as $k = \arg\max_i \left[g_i + \log\left(\pi_i\right)\right]$, and subsequently select the legal orientation as $\theta_c = o_k$.

Algorithm 2: Cypress— Bilevel Placement Optimization

1. Create an initial placement parameterized by $(\mathbf{x}, \mathbf{y}, \boldsymbol{\pi})$
2. Obtain the global placement loss function $\mathcal{L}$ defined in (8).
3. **while** *not converged* **do**
4. 1. Update positions $\mathbf{x}, \mathbf{y}$ by descending $\nabla\mathcal{L}$ while freezing $\boldsymbol{\pi}$, using $\mathcal{P}_{\mathbf{x},\mathbf{y}}$ as gradient preconditioner;
5. 2. Update class probabilities $\boldsymbol{\pi}$ by descending $\nabla\mathcal{L}$ while freezing $\mathbf{x}, \mathbf{y}$, using $\mathcal{P}_{\boldsymbol{\pi}}$ as gradient preconditioner;
6. Derive final placement by $(\mathbf{x}, \mathbf{y}, \boldsymbol{\pi})$
7. Legalize placement.

With the continuous relaxation, the optimization problem becomes jointly optimizing the component coordinates $(\mathbf{x}, \mathbf{y})$ and the component orientations $\boldsymbol{\theta}$. Analogous to neural architecture search using gradient descent [29], we apply a bilevel optimization process [9], described in Algorithm 2.

4.2.3 Density. We follow the density model of DREAMPlace [27] which models components as electric charges, density as potential energy, and density gradient as electric field. The electric potential and field distribution is obtained by solving Poisson's equation from the charge density distribution via spectral methods with a two-dimensional fast Fourier transform (FFT). We define the density as $\mathrm{D}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta})$. Our modifications are specific to the PCB domain: (1) The inclusion of $\boldsymbol{\theta}$ remains consistent with the DREAMPlace formulation, as components are first rotated, which potentially swaps the height and width of a component when calculating the density map, (2) Additionally, to account for the two-sided nature of PCBs, we construct separate density maps for each side, ensuring an accurate representation of the overall density. Note that in our formulation, the side assigned to each component is predetermined by the designer's choice, and we leave as future work its optimization.

4.2.4 Cost Function. To put everything together, the following equation is the combined cost function we solve with gradient descent:

$$\mathcal{L}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) = \mathrm{WL}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) + \lambda_D \, \mathrm{D}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) + \lambda_{NC} \, \mathrm{NC}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) \tag{8}$$

where λ_D, λ_{NC} are the weights of the density and the net crossing, respectively. At the beginning of the placement process, all components are positioned at an initial starting point, minimizing wirelength due to overlap. We then progressively increase the Lagrange multiplier λ_D for the density term to spread the components and reach a minimal energy state at the end of global placement. We adopt the dynamic density schedule introduced in DREAMPlace 3.0 [14], where the density weight is exponentially increased, controlled by a density factor bounded within predefined lower and upper limits. Simultaneously, the weight of the net crossing term, λ_{NC}, gradually increases to reduce net crossings as components are distributed. We apply a static linear schedule for the net crossing weight: $\lambda_{NC}^{t+1} = \lambda_{NC}^{t} + \eta$, with η being a tunable parameter.

4.3 Preconditioning

We observed that the varying sizes of PCB components can lead to slow convergence or even divergence during optimization. To address this, we apply divergence-aware gradient preconditioning [14]. The second-order derivative of the wirelength objective is estimated based on the pin count of each component [32], while the second-order derivative of the density term is approximated by the component area [27, 32], forming the diagonal of the Hessian matrix. We found no preconditioning term was needed in practice for the net crossing, given that the wirelength preconditioning term can account for it. The precondition operator for updating the positions $\mathbf{x}$, $\mathbf{y}$ is then defined as follows:

$$\begin{aligned}\mathcal{P}_{\mathbf{x},\mathbf{y}} &= \min\left(1, \left(\nabla^2 \mathrm{WL}(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}) + \beta\lambda_D \nabla^2 \mathrm{D}\left(\mathbf{x}, \mathbf{y}, \boldsymbol{\theta}\right)\right)^{-1}\right) \\ &= \min\left(1, \left(\#\,\mathrm{pin}(\mathbf{c}) + \beta\lambda_D \,\mathrm{area}(\mathbf{c})\right)^{-1}\right)\end{aligned} \tag{9}$$

where $\#\,\mathrm{pin}(\cdot)$ is the number of pins of a component, and $\mathrm{area}(\cdot)$ its area. We initialize β to 1 and linearly increase it with each iteration. This gradual increase in β reduces the movement of larger

components in the later stages of the optimization process. For orientation updates, since class probabilities are only updated by gradients from wirelength and net crossings, we omit the second term of the preconditioning operation for the density calculation:

$$\mathcal{P}_{\boldsymbol{\pi}} = \min\left(1, \#\,\mathrm{pin}(\mathbf{c})^{-1}\right). \tag{10}$$

4.4 Legalization

We build upon the approach used in DREAMPlace [27] to implement legalization as an operation. Unlike DREAMPlace, our method legalizes the two PCB sides separately, ensuring that components on the same side do not overlap. The process begins with a Tetris-like procedure inspired by NTUplace3 [5], followed by Abacus row-based legalization [39]. The process begins with a Tetris-like procedure inspired by NTUplace3 [5], followed by Abacus row-based legalization [39]. This step is performed after transferring placement from GPU to CPU, where legalization is executed entirely on a single CPU thread in just a few seconds, even for large designs with tens of thousands of components.

4.5 Constraint Handling

Cypress supports two types of constraints commonly encountered in PCB design: spacing constraints, which enforce a minimum distance between components, and fixed-component constraints, which account for pre-placed or immovable components.

4.5.1 Spacing Constraints. To enforce spacing constraints, Cypress employs the macro halo technique introduced in AutoDMP [1]. This technique temporarily enlarges the effective footprint of each component during placement, creating a virtual buffer that prevents overlap. Once placement and legalization are complete, the halos are removed, and the components revert to their original sizes while maintaining the required spacing.

4.5.2 Fixed Components. In many PCB designs, certain components are pre-placed and must remain fixed due to mechanical constraints or performance requirements, such as high-speed or high-current elements. For fixed components, Cypress takes their positions and orientations into account when calculating the wirelength, density, and net crossing, as well as during legalization, while their positions and orientations are held constant by excluding them from the parameter update steps. To mitigate the potential routing challenges posed by fixed components, Cypress optimizes the placement of movable components to improve routability and minimize wirelength by considering the positions of fixed components in the loss function. This ensures that movable components adapt to the constraints imposed by pre-placed obstacles, achieving high routability and short wirelength.

For components with partial constraints, Cypress allows selective updates. For instance, the optimization can fix the position $(\mathbf{x}, \mathbf{y})$ while adjusting the orientation $\boldsymbol{\pi}$, or vice versa. This flexibility is achieved by masking updates to specific parameters, enabling fine-grained control over placement adjustments.

4.6 Design Space Exploration

DREAMPlace-based placers enable placement optimization with the latest gradient optimizers, such as Adam and Nesterov. However, this opens up a search space for the optimizer parameters, such as learning rate, weight decay, etc. In addition, the initial placement location also affects the final solution quality. Therefore, we follow AutoDMP [1] to use multi-objective Bayesian optimization (MOTPE) for parameter space exploration. Table 2 lists the parameter space we will tune in our experiments.

Table 2: Parameter space of Cypress

Parameter	Search Range
horiz. initial position	[0.2, 0.8](%)
vertical initial position	[0.2, 0.8](%)
init. density weight	[1e-6, 1.0]
target density	[0.1, 1.0]
init. net crossing weight	[1e-6, 1.0]
LogSumExp init. γ	[0.1, 0.5]
init. learning rate	[1e-4, 1e-2]
optimizer	{Adam, Nesterov}

The design space exploration process can run in parallel across multiple GPUs. The MOTPE algorithm searches for Pareto-optimal points along the axes of wirelength (HPWL), net crossing, and density, effectively exploring the trade-offs between multiple objectives to produce placements with high routability.

5 PCB Placement Benchmark

We develop a PCB benchmark suite based on realistic commercial designs to facilitate comparison between PCB placement tools and establish a meaningful baseline. The suite consists of two sets: a small open-source set containing PCB designs with 41 to 476 components, and a larger set with 5,118 to 6,628 components. Detailed statistics are provided in Table 3, #movable refers to the number of movable components.

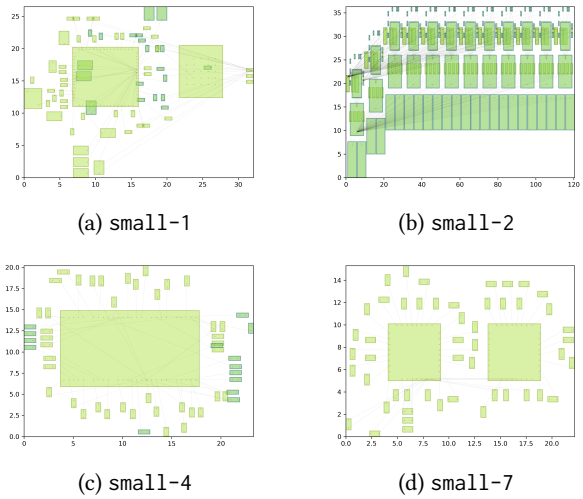
5.1 Benchmark Synthesis Method

Existing PCB placement tools have limited scalability. For example, Quilter has a 95% success rate for designs with less than 500 pins and 100 components [37]. Therefore, we need to synthesize a benchmark suite of designs with 10s–100s of components to facilitate meaningful comparison between tools. Additionally, we consider how PCB engineers approach placement for larger designs. They often start by performing local placement on a subset of components off-canvas, then position the group on the main board. These component groups are typically chosen based on functionality, such as placing a GPIO expander alongside its connected components. Such practice makes small but realistic test cases particularly important for industry.

Based on this observation, we synthesize 10 small PCB designs `small-1` to `small-10`. The benchmark data includes component shapes, layer assignment, pin positions, net connections. These designs are based on functional group of commercial PCB designs, including Ethernet IC controller, power controller, GPIO expander, CPLDs. Fig. 4 shows four examples of small benchmarks. Light

Table 3: PCB Benchmark Suite Statistics

	#components	#movable	#pins	#nets
small-1	57	57	167	85
small-2	240	240	612	84
small-3	63	63	156	51
small-4	43	43	129	82
small-5	41	41	105	49
small-6	46	46	85	26
small-7	50	50	153	95
small-8	115	115	5334	199
small-9	476	476	2212	1513
small-10	136	136	582	403
big-1	6589	1714	18140	5620
big-2	6537	1918	21824	7190
big-3	5118	1293	11618	2914
big-4	6542	1922	21893	7166
big-5	6628	2152	20838	6937



(a) small-1 (b) small-2

(c) small-4 (d) small-7

Figure 4: Examples from the small PCB benchmark suite.

green indicates components on the top layer, deep green represents components on the bottom layer, and black lines depict connections between pins.

5.2 Proprietary Benchmarks

To evaluate the scalability of our approach, we included five large benchmarks big-1 to big-5 based on real commercial PCB designs. While these designs will not be part of the open-source benchmark, they are used exclusively for evaluating our tool. These fully realized commercial designs feature pre-placed VDD, high-speed signals, and ASICs. It is important to note that existing PCB placement tools cannot handle designs of this scale, highlighting the challenges our approach seeks to address.

5.3 PCB Intermediate Representation

PCB data formats are fragmented across different EDA tools. For instance, KiCAD uses its own formats for PCB and schematics [22], some placement tools rely on the Bookshelf format [35], and autorouters like Freerouting accept the Specctra DSN format [12]. To address this fragmentation, we develop and open-source a YAML-based PCB intermediate representation (IR) along with translation tools to interface with KiCAD, IDF (Intermediate Dataformat), and Bookshelf formats, as well as visualization toolkits. The PCB IR captures component, pin, and net information in a format that is portable, extensible, and easy to visualize, facilitating cross-platform compatibility.

6 Experiments

The Cypress framework builds upon AutoDMP [1] and DREAMPlace [14, 25, 27, 28], with PCB-specific cost functions implemented as CUDA kernels and custom PyTorch operators. We evaluate Cypress on the proposed benchmarks, comparing it against three baseline tools from academia, the open-source community, and commercial solutions. To assess the scalability of Cypress, we also test it on real commercial designs. The three baseline tools are: (1) NS-Place [7], a state-of-the-art academic PCB placement tool that uses a net separation formulation to optimize routability by separating the convex hull of nets. It applies mixed-integer linear programming (MILP) for legalization. We obtain the NS-Place implementation source code from GitHub[2]. Furthermore, we introduce an enhanced NS-Place baseline, termed NS-Place+, which replaces the MILP legalization with Cypress's legalization algorithm described in Section 4.4, as MILP demonstrates suboptimal performance. (2) SA-PCB [10], an open-source tool from OpenROAD based on simulated annealing which supports double-sided designs and component rotation. (3) Quilter [37], a commercial solution that employs reinforcement learning to automate PCB placement and routing. We evaluate the placement solutions using several key metrics: wirelength (HPWL), net crossing, routability, and routed track length. To assess routability and routed track length, we utilize FreeRouting v1.9.0 [12], an open-source PCB autorouter. For all designs, we assume the board has two free copper layers for routing. Routability is defined as the ratio of successfully routed pin pairs to the total number of pin pairs reported by FreeRouting.

The reported run times are based on the following setup: Cypress was executed on an NVIDIA V100 GPU, while NS-Place and SA-PCB were run on an Intel Xeon 2.2GHz CPU with 80GB memory. Quilter does not run locally; designs in kicad_pcb format were submitted to Quilter's cloud service and proprietary server infrastructure. The Quilter experiments were conducted using its April 04, 2024 release with the following settings: two-sided PCB and no pre-placed components, all other configurations are set to default.

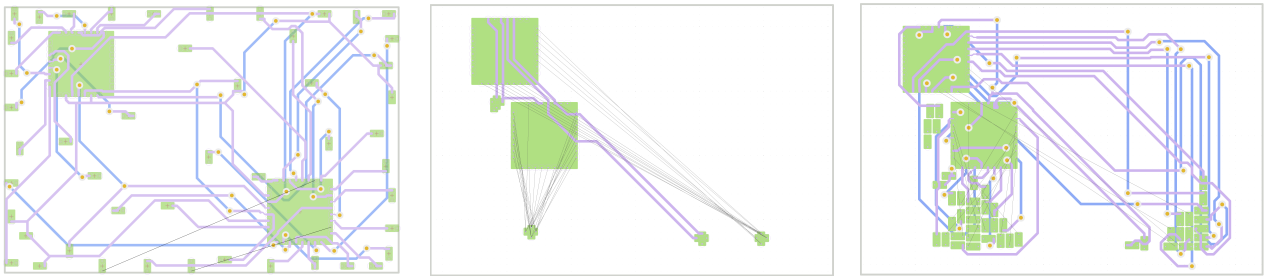
6.1 Small Benchmark Results

In addition to the baseline placement tools, we include PCB engineer placement results as a human baseline. It is important to note that the human baseline is derived from a larger design, where additional constraints, such as thermal considerations, are factored in. As a result, the placement may be more conservative in accounting for

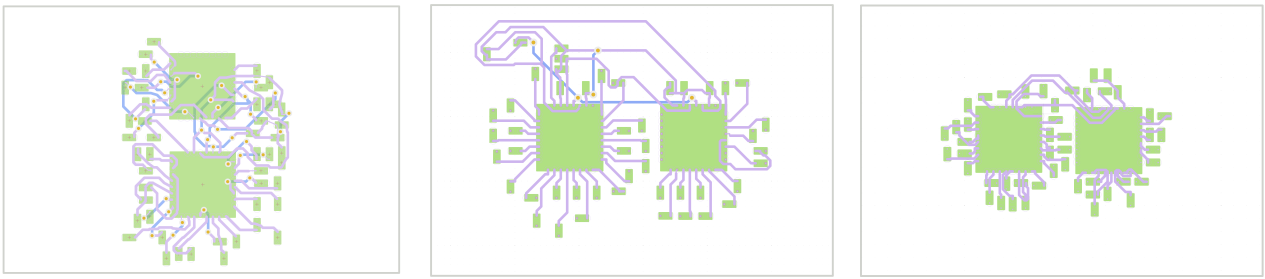
[2] https://github.com/choltz95/pcb-placement

Table 4: Results of the small benchmark suite — HPWL denotes half-perimeter wirelength (lower is better), NC represents net crossings (lower is better), RB indicates routability (higher is better), and TL denotes routed track length (lower is better). Averages are normalized relative to SA-PCB.

	SA-PCB		NS-Place		NS-Place+		Quilter		Human		Cypress	
	HPWL / NC	RB / TL	HPWL / NC	RB / TL	HPWL / NC	RB / TL	HPWL / NC	RB / TL	HPWL / NC	RB / TL	HPWL / NC	RB / TL
S1	18.2K / 122.3	100.0% / 1.4K	7.9K / 82.1	60.8% / 0.4K	7.4K / 50.3	100.0% / 0.8K	2.3K / 137.4	100.0% / 0.3K	4.0K / 74.6	100.0% / 0.3K	1.6K / 46.5	100.0% / 0.2K
S2	92.6K / 172.9	100.0% / 9.0K	15.4K / 6.0	54.9% / 0.6K	60.8K / 1643.1	99.6% / 6.5K	26.2K / 79.3	100.0% / 3.2K	11.4K / 21.8	100.0% / 1.2K	6.5K / 36.1	100.0% / 1.1K
S3	11.0K / 81.0	99.0% / 1.2K	3.3K / 60.4	93.0% / 0.7K	6.7K / 96.9	100.0% / 0.8K	2.8K / 42.8	100.0% / 0.2K	3.7K / 111.0	100.0% / 0.2K	1.7K / 52.9	100.0% / 0.2K
S4	12.9K / 8.2	100.0% / 0.9K	4.8K / 0.6	17.8% / 0.1K	5.8K / 11.1	77.5% / 0.5K	4.1K / 10.0	100.0% / 0.2K	4.5K / 5.1	100.0% / 0.3K	1.8K / 6.2	100.0% / 0.2K
S5	6.0K / 23.4	91.0% / 0.6K	2.3K / 16.9	71.4% / 0.3K	6.2K / 46.8	80.4% / 0.5K	2.7K / 27.2	100.0% / 0.2K	2.3K / 17.8	100.0% / 0.2K	1.5K / 20.1	100.0% / 0.2K
S6	5.5K / 56.2	98.3% / 0.5K	3.5K / 31.3	86.4% / 0.5K	3.3K / 55.3	88.1% / 0.4K	2.0K / 41.9	100.0% / 0.1K	1.4K / 36.0	100.0% / 0.1K	0.9K / 18.4	100.0% / 0.1K
S7	11.4K / 20.7	96.6% / 0.6K	6.9K / 10.5	17.0% / 0.1K	7.2K / 17.3	61.7% / 0.6K	2.6K / 13.7	100.0% / 0.2K	2.9K / 4.5	100.0% / 0.2K	0.7K / 18.7	100.0% / 0.1K
S8	118.3K / 34.0	82.9% / 8.4K	44.0K / 31.5	63.1% / 2.7K	52.0K / 119.3	87.2% / 5.1K	21.2K / 25.6	100.0% / 2.1K	7.1K / 29.6	98.5% / 0.4K	3.7K / 9.7	100.0% / 0.4K
S9	1322.3K / 2164.7	98.6% / 23.8K	- / -	-% / -	- / -	-% / -	- / -	-% / -	161.5K / 1911.8	90.4% / 7.0K	25.0K / 809.3	100.0% / 3.5K
S10	131.1K / 123.7	81.5% / 7.8K	66.6K / 105.9	27.1% / 1.1K	113.1K / 157.0	86.2% / 8.5K	25.9K / 102.0	100.0% / 2.0K	16.4K / 164.6	100.0% / 1.0K	8.4K / 84.6	100.0% / 0.5K
Avg	1.0 / 1.0	1.0 / 1.0	0.38 / 0.54	0.58 / 0.21	0.64 / 3.42	0.92 / 0.78	0.22 / 0.75	1.06 / 0.28	0.13 / 0.72	1.06 / 0.13	0.07 / 0.46	1.06 / 0.1



(a) SA-PCB (97% routable) (b) NS-Place (17% routable) (c) NS-Place+ (62% routable)



(d) Quilter (100% routable) (e) Human (100% routable) (f) Cypress (100% routable)

Figure 5: Routed PCB design of benchmark `small-7` placed by SA-PCB, NS-Place, NS-Place+, Quilter, Human, and Cypress. All components are movable. Unrouted pin pairs are shown as black line segments.

nearby components, which explains why other tools achieve better wirelength performance compared to the human baseline.

Table 4 shows the PCB placement results on 10 small benchmarks. Note that net crossings are not integers because we employ the continuous function definition given in Eq. 4. We performed parameter space exploration using MOTPE for 30 iterations, using four NVIDIA V100 GPUs in parallel. The DSE for small benchmarks took between 3.3 to 29.4 minutes, producing a Pareto frontier. We selected the design with the lowest HPWL as the final design. Cypress outperforms baseline tools across several metrics. Against SA-PCB, Cypress improves HPWL by 75% to 98.1%, averaging 89.5%, and net crossing by 45.7% on average, with a maximum of 79.1%. Routability improves by 1.0–1.2×, and track length is 3.6–19.7× shorter. Compared to NS-Place, Cypress improves HPWL by 69.3% on average, and net crossing varies from a 963% degradation to a 69.4% improvement. Routability improves by 2.7×. Cypress achieves an average HPWL improvement of 81.5% over NS-Place+, with net crossing ranging from an 8.4% degradation to a 97.8% improvement. Routability increases by 1.2×. Against Quilter, Cypress improves HPWL by 57.8% on average, and net crossing by 57.8%. Routability remains similar, except for S9, where Quilter failed and Cypress produced a fully routable design. Compared to human designers, Cypress improves HPWL by 34.4–84.5%, with an average of 54.2%, while routability remains the same and track length decreases by 1.5×.

In terms of routability, NS-Place performs poorly due to unresolved overlaps caused by legalization failures. NS-Place employs MILP with the Coin-or branch and cut (CBC) solver backend [11], with a solver time-out limit of 60 minutes. The solver timed out for benchmarks S1, S2, S4, S5, S6, S7, S8, and S10. To enable a fair comparison with Cypress, we introduced an additional baseline,

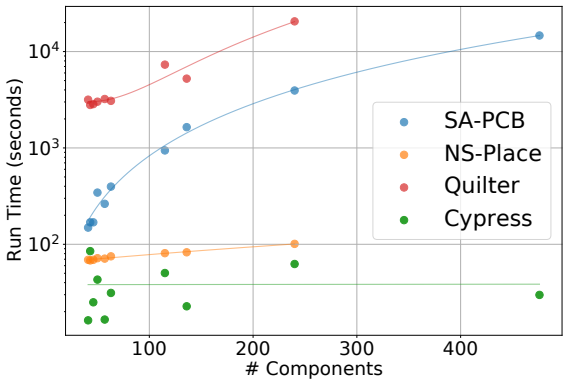


Figure 6: Scalability Comparison

NS-Place+, which replaces the MILP legalizer with Cypress's legalization algorithm. After resolving overlaps, NS-Place+ exhibits a 1.7× higher wirelength and 6.3× higher net crossing. Nevertheless, NS-Place+ achieves a 1.6× improvement in routability due to successful legalization.

6.1.1 Scalability. Fig. 6 illustrates the relationship between the run time and the number of components in the design. The run time of SA-PCB increases linearly with component count, while Quilter's grows exponentially. Quilter's longer run time can be attributed to two factors: its reinforcement learning approach, which relies on a sample learning process, and the fact that it includes routing within the optimization cycle. For the NS-Place group, we exclude the MILP legalization time and report only the net convex hull separation optimization time. This decision is due to the frequent timeouts of the NS-Place MILP solver, whereas NS-Place+ utilizes Cypress's legalizer, which complets in less than one second. In contrast, Cypress exhibits strong scalability, largely due to its GPU acceleration. It is important to note that the reported run time in Fig. 6 for Cypress reflects the time for a single placement job, not the full DSE run time.

Cypress's DSE leverages parallel workers to evaluate multiple design candidates in parallel across multiple GPUs, enabling faster exploration by scaling the number of GPUs. Using four NVIDIA V100 GPUs with four workers in parallel, the DSE for benchmarks S1, S3, S4, S5, S6, S7, and S10 completes in 200.5 to 350.7 seconds, S9 requires 584.5 seconds, and S2 takes 1764.7 seconds.

6.1.2 Results Analysis. To compare the results of each tool, we use the `small-7` design as an example. Figure 5 illustrates the routed PCB designs produced by SA-PCB, NS-Place, NS-Place+, Quilter, manual results, and Cypress. We analyze the strengths and limitations of each baseline as follows:

- SA-PCB focuses on avoiding component overlap and minimizing wirelength, but due to an improper balance between these objectives, it often pushes components to the borders. While this improves routability for smaller designs, this approach can degrade routability on a larger and denser design like `small-9`.
- NS-Place optimizes routability by minimizing the overlaps between the net convex hulls, a very conservative but less precise model than net crossing, as discussed in Section 4.2.1. Although this is very efficient for avoiding net crossings, this overconstraining without proper balance with density and wirelength leads to suboptimal solutions where small components cluster into non-overlapping convex hulls, causing blockages and severely reducing routability. Furthermore, NS-Place depends heavily on legalization to resolve overlaps, leading to substantial deviations between global and detailed placement stages, thus producing unpredictable results. The MILP-based legalization approach, moreover, lacks scalability and frequently fails, leaving the final solution with unresolved overlaps.
- NS-Place+ incorporates Cypress's legalization algorithm in place of MILP, effectively resolving overlaps with a runtime of less than one second. Consequently, eliminating overlaps results in increased wirelength. Additionally, we observe a degradation in net crossing after overlap removal, indicating that NS-Place's over-constrained net convex hull separation contributes to overlap issues. While overlap removal improves routability, NS-Place still fails to achieve 100% routability in many cases.
- Quilter, a closed-source commercial placer, leverages reinforcement learning and integrates a router in the optimization loop, yielding highly routable designs. However, this comes at the cost of long run times and limited scalability.
- The human baseline is extracted from a larger design. Consequently, the placement tends to be more conservative, accounting for nearby components, which explains why Cypress achieves better wirelength performance than the human baseline.

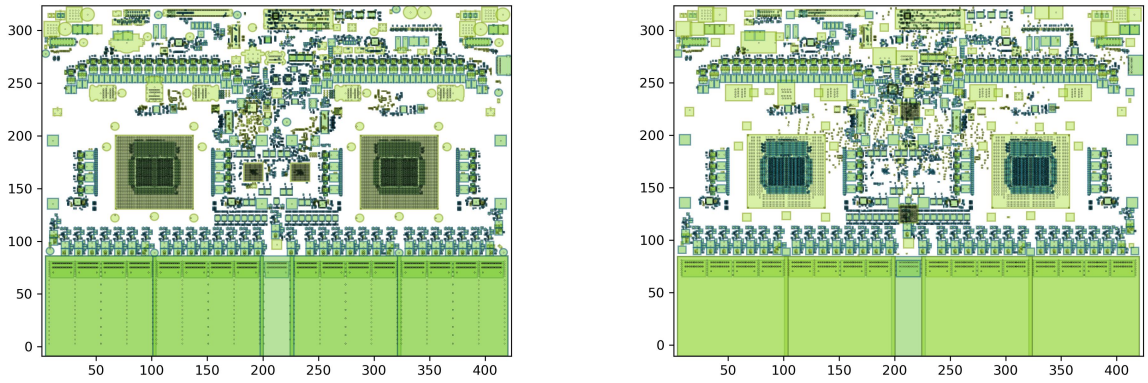
Finally, Cypress highlights the significance of multi-objective optimization. By considering wirelength, net crossing, and density simultaneously, Cypress consistently produces highly routable designs with low wirelength.

6.2 Big Benchmark Results

We compare Cypress under various configurations on commercial designs. A direct comparison with fully human-designed layouts is avoided, as human designers take into account a much broader set of constraints, making such a comparison less meaningful. Instead, this section is framed as an ablation study.

Table 5 presents the results of Cypress on five large benchmark circuits (B1–B5), showing HPWL and net crossing (NC) under three configurations: wirelength and density optimization, the addition of net crossing optimization, and the inclusion of orientation-aware placement. We used the same parameter space exploration setup as for the small benchmark, with runtime ranging from 143 to 318 minutes.

In B1, Cypress starts with HPWL and net crossings (NC) of 2.87M and 100.7K, respectively. Adding net crossing minimization reduces NC to 31.4K, with HPWL increasing slightly to 3.04M. When orientation awareness is applied, HPWL returns to 2.87M, and NC drops further to 15.3K. For B2, Cypress produces 6.53M HPWL and 84.1K NC initially. After net crossing minimization, NC reduces significantly to 15.6K without affecting HPWL. Adding orientation awareness reduces NC slightly to 15.3K, while HPWL remains unchanged. In B3, the initial configuration yields 1.33M



(a) Human (b) Cypress

Figure 7: Placement result comparison of benchmark `big-1`: human v.s. Cypress. Green boxes represent components on the top layer, blue boxes represent components on the bottom layer. Nets are omitted for clarity.

Table 5: Big benchmark suite results – HPWL denotes half-perimeter wirelength (lower is better), NC stands for net crossing (lower is better). Averages are normalized relative to WL+Density.

	WL + Density		+ Net Crossing		+ Orientation	
	HPWL	NC	HPWL	NC	HPWL	NC
B1	2.87M	100.7K	3.04M	31.4K	2.87M	15.3K
B2	6.53M	84.1K	6.53M	15.6K	6.53M	15.3K
B3	1.33M	91.5K	1.36M	33.5K	1.33M	37.4K
B4	6.58M	78.9K	7.20M	9.7K	6.57M	16.8K
B5	5.24M	89.9K	7.25M	46.4K	6.28M	34.0K
Avg	1.0	1.0	1.13	0.31	1.05	0.27

HPWL and 91.5K NC. Net crossing minimization lowers NC to 33.5K, with a minor HPWL increase to 1.36M. However, orientation awareness slightly increases NC to 37.4K, while HPWL returns to 1.33M. For B4, Cypress starts with 6.58M HPWL and 78.9K NC. Net crossing minimization reduces NC to 9.7K, though HPWL rises to 7.20M. With orientation awareness, HPWL drops to 6.57M, while NC increases slightly to 16.8K. Finally, in B5, the initial setup yields 5.24M HPWL and 89.9K NC. Minimizing net crossings reduces NC to 46.4K but raises HPWL to 7.25M. Adding orientation awareness improves HPWL to 6.28M and reduces NC further to 34.0K. We observe similar trends in B1 through B5, where minimizing net crossings increases wirelength, but adding orientation awareness reduces HPWL again, resulting in an overall improved design.

Fig. 7 compares the human-placed large design `big-1` with the version generated by Cypress. We fixed critical components, such as ASICs and power supply units, and allowed Cypress to place the noncritical components, subject only to spacing constraints. Green blocks represent components on the top layer, while blue blocks represent components on the bottom layer. Cypress uses bounding boxes to represent components, as shown in Fig. 7(b). Overall, we observe similarities in clustering smaller components between the human and Cypress placements. A notable difference appears with the two central CPLDs: in the human design, they are placed horizontally, while in the Cypress placement—guided by wirelength, density, and net crossing—they are placed vertically, though still in a symmetrical arrangement. Additionally, smaller components have been pushed into the open spaces. Since we did not apply keep out area constraints, this behavior is expected, although it may not occur in real-world designs.

7 Limitations

Cypress supports spacing constraints during PCB placement but does not yet handle height or alignment constraints. Height constraints (e.g., "keep-out" areas) prevent tall components from certain regions, while alignment constraints require specific components to be aligned. These constraints can be considered through penalties in the objective function or through legalization, and preprocessing can merge aligned components into single units during placement.

Cypress also does not handle critical components such as high-current paths or high-speed signals. Instead, it focuses on fast placement for noncritical components, accelerating the iterative design process. Constraints like thermal, mechanical, and electromagnetic factors are not formally encoded and are left for future work.

8 Conclusion

In this work, we presented a scalable, GPU-accelerated PCB placement method inspired by VLSI techniques, addressing the challenges of flexible design spaces and routing constraints. Our approach outperforms state-of-the-art tools in routability, track length, and run time, demonstrating superior scalability for large commercial designs. We also introduced a synthesized benchmark suite to support tool comparisons and track progress. The code and benchmarks are open-sourced to advance research in PCB placement.

ACKNOWLEDGEMENTS – We gratefully acknowledge the anonymous reviewers for their valuable feedback. We also thank Yi-Chen Lu, Chia-Tung Ho, Mark Kilgard at NVIDIA for their insightful feedback on the initial version of the Cypress framework. This work is supported in part by NSF Award #2212371.

References

[1] Anthony Agnesina, Puranjay Rajvanshi, Tian Yang, Geraldo Pradipta, Austin Jiao, Ben Keller, Brucek Khailany, and Haoxing Ren. 2023. AutoDMP: Automated DREAMPlace-based Macro Placement. In *Proceedings of International Symposium on Physical Design*. Association for Computing Machinery, New York, NY, USA, 149–157.

[2] Riccardo Cecchetti, Francesco de Paulis, Carlo Olivieri, Antonio Orlandi, and Markus Buecker. 2020. Effective PCB Decoupling Optimization by Combining an Iterative Genetic Algorithm and Machine Learning. *Electronics* 9, 8 (2020), 1243.

[3] Tony F Chan, Jason Cong, Joseph R Shinnerl, Kenton Sze, and Min Xie. 2006. mPL6: Enhanced Multilevel Mixed-Size Placement. In *Proceedings of International Symposium on Physical Design*. Association for Computing Machinery, New York, NY, USA, 212–214.

[4] Tung-Chieh Chen, Tien-Chang Hsu, Zhe-Wei Jiang, and Yao-Wen Chang. 2005. NTUplace: A Ratio Partitioning Based Placement Algorithm for Large-Scale Mixed-Size Designs. In *Proceedings of International Symposium on Physical Design*. Association for Computing Machinery, New York, NY, USA, 236–238.

[5] Tung-Chieh Chen, Zhe-Wei Jiang, Tien-Chang Hsu, Hsin-Chen Chen, and Yao-Wen Chang. 2007. NTUplace3: An Analytical Placer for Large-Scale Mixed-Size Designs. In *Modern Circuit Placement: Best Practices and Results*. Springer, New York, NY, USA, 289–309.

[6] Yee-Ming Chen and Chun-Ta Lin. 2007. A Particle Swarm Optimization Approach to Optimize Component Placement in Printed Circuit Board Assembly. *The International Journal of Advanced Manufacturing Technology* 35 (2007), 610–620.

[7] Chung-Kuan Cheng, Chia-Tung Ho, and Chester Holtz. 2022. Net Separation-Oriented Printed Circuit Board Placement via Margin Maximization. In *27th Asia and South Pacific Design Automation Conference*. IEEE, New York, NY, USA, 288–293.

[8] Chung-Kuan Cheng, Andrew B Kahng, Ilgweon Kang, and Lutong Wang. 2018. RePlAce: Advancing Solution Quality and Routability Validation in Global Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 9 (2018), 1717–1730.

[9] Benoît Colson, Patrice Marcotte, and Gilles Savard. 2007. An Overview of Bilevel Optimization. *Annals of Operations Research* 153 (2007), 235–256.

[10] The OpenROAD Project Contributors. 2024. SA-PCB: Simulated Annealing for Printed Circuit Board Placement. https://github.com/The-OpenROAD-Project/SA-PCB.

[11] John J. Forrest and the COIN-OR contributors. 2025. COIN-OR Branch-and-Cut Solver (Cbc). https://github.com/coin-or/Cbc. Accessed: 2025-01-03.

[12] Freerouting Project. 2024. Freerouting: A Free Autorouter. https://github.com/freerouting/freerouting Accessed: 2024-09-27.

[13] Charles N Frisbee. 1996. An Overview of Placement and Routing Algorithms for PCB, VLSI and MCM Designs with a Proposal for a new MCM Routing Algorithm. *University of Arkansas*. 1 (1996), 1–51.

[14] Jiaqi Gu, Zixuan Jiang, Yibo Lin, and David Z Pan. 2020. DREAMPlace 3.0: Multi-Electrostatics based robust VLSI placement with region constraints. In *Proceedings of International Conference on Computer-Aided Design*. Association for Computing Machinery, New York, NY, USA, 1–9.

[15] Zheming Gu, Ling Zhang, Hang Jin, Tuomin Tao, Da Li, and Er-Ping Li. 2022. Deep Reinforcement Learning-Based Ground-Via Placement Optimization for EMI Mitigation. *IEEE Transactions on Electromagnetic Compatibility* 65, 2 (2022), 564–573.

[16] Shinn-Ying Ho, Shinn-Jang Ho, Yi-Kuang Lin, and WC-C Chu. 2004. An Orthogonal Simulated Annealing Algorithm for Large Floorplanning Problems. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 12, 8 (2004), 874–877.

[17] William Ho and Ping Ji. 2005. A genetic algorithm to optimise the component placement process in PCB assembly. *The International Journal of Advanced Manufacturing Technology* 26 (2005), 1397–1401.

[18] Meng-Kai Hsu and Yao-Wen Chang. 2012. Unified Analytical Global Placement for Large-Scale Mixed-Size Circuit Designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 9 (2012), 1366–1378.

[19] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical Reparameterization with Gumbel-Softmax. *arXiv preprint arXiv:1611.01144* 1 (2016), 1–13.

[20] Zhe-Wei Jiang, Tung-Chieh Cheny, Tien-Chang Hsuy, Hsin-Chen Chenz, and Yao-Wen Changyz. 2006. NTUplace2: A Hybrid Placer using Partitioning and Analytical Techniques. In *Proceedings of International Symposium on Physical Design*. Association for Computing Machinery, New York, NY, USA, 215–217.

[21] Cooper Jones. 2023. *Distributed Monte Carlo Tree Search With Applications To Chip Design*. Ph. D. Dissertation. Massachusetts Institute of Technology.

[22] GR Kanagachidambaresan. 2021. Introduction to KiCad Design for Breakout and Circuit Designs. In *Role of Single Board Computers (SBCs) in Rapid IoT Prototyping*. Springer, New York, NY, USA, 165–175.

[23] LP Khoo and TK Ng. 1998. A Genetic Algorithm-Based Planning System for PCB Component Placement. *International Journal of Production Economics* 54, 3 (1998), 321–332.

[24] Samuel P Kuo. 2006. *PCB Design and Simulation Using Cadence Allegro 15.5*. Ph. D. Dissertation. University of Illinois at Urbana-Champaign.

[25] Peiyu Liao, Dawei Guo, Zizheng Guo, Siting Liu, Yibo Lin, and Bei Yu. 2023. DREAMPlace 4.0: Timing-Driven Placement with Momentum-Based Net Weighting and Lagrangian-Based Refinement. *Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 10 (2023), 3374–3387.

[26] Jai-Ming Lin, Tsung-Chun Tsai, and Rui-Ting Shen. 2023. Routability-Driven Orientation-Aware Analytical Placement for System in Package. In *2023 IEEE/ACM International Conference on Computer Aided Design*. IEEE, New York, NY, USA, 1–8.

[27] Yibo Lin, Shounak Dhar, Wuxi Li, Haoxing Ren, Brucek Khailany, and David Z Pan. 2019. DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement. In *Proceedings of Annual Design Automation Conference*. IEEE, New York, NY, USA, 1–6.

[28] Yibo Lin, David Z Pan, Haoxing Ren, and Brucek Khailany. 2020. DREAMPlace 2.0: Open-Source GPU-Accelerated Global and Detailed Placement for Large-Scale VLSI Designs. In *China Semiconductor Technology International Conference*. IEEE, New York, NY, USA, 1–4.

[29] Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. DARTS: Differentiable Architecture Search. *arXiv preprint arXiv:1806.09055* 1 (2018), 1–13.

[30] Wen-Hao Liu, Anthony Agnesina, and Haoxing Mark Ren. 2024. Challenges for Automating PCB Layout. In *Proceedings of International Symposium on Physical Design*. Association for Computing Machinery, New York, NY, USA, 91–92. doi:10.1145/3626184.3635285

[31] Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis J-H Huang, Chin-Chi Teng, and Chung-Kuan Cheng. 2014. ePlace: Electrostatics-Based Placement Using Nesterov's Method. In *Proceedings of Annual Design Automation Conference*. Association for Computing Machinery, New York, NY, USA, 1–6.

[32] Jingwei Lu, Pengwen Chen, Chin-Chih Chang, Lu Sha, Dennis Jen-Hsin Huang, Chin-Chi Teng, and Chung-Kuan Cheng. 2015. ePlace: Electrostatics-Based Placement Using Fast Fourier Transform and Nesterov's Method. *Transactions on Design Automation of Electronic Systems* 20, 2 (2015), 1–34.

[33] Jingwei Lu, Hao Zhuang, Pengwen Chen, Hongliang Chang, Chin-Chih Chang, Yiu-Chung Wong, Lu Sha, Dennis Huang, Yufeng Luo, Chin-Chi Teng, et al. 2015. ePlace-MS: Electrostatics-Based Placement for Mixed-Size Circuits. *Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34, 5 (2015), 685–698.

[34] Hiroshi Murata, Kunihiro Fujiyoshi, and Mineo Kaneko. 1997. VLSI/PCB Placement with Obstacles Based on Sequence-Pair. In *Proceedings of International Symposium on Physical Design*. Association for Computing Machinery, New York, NY, USA, 26–31.

[35] Gi-Joon Nam, Charles J Alpert, Paul Villarrubia, Bruce Winter, and Mehmet Yildiz. 2005. The ISPD2005 Placement Contest and Benchmark Suite. In *Proceedings of International Symposium on Physical Design*. Association for Computing Machinery, New York, NY, USA, 216–220.

[36] William C Naylor, Ross Donelly, and Lu Sha. 2001. Non-Linear Optimization System and Method for Wire Length and Delay Optimization for an Automatic Electric Circuit Placer. US Patent 6,301,693.

[37] Quilter. 2024. Quilter Technology: Automated Circuit Board Design. https://www.quilter.ai/technology. Accessed: 2024-09-26.

[38] Peter Spindler and Frank M Johannes. 2007. Fast and Accurate Routing Demand Estimation for Efficient Routability-Driven Placement. In *Design, Automation & Test in Europe Conference & Exhibition*. IEEE, Association for Computing Machinery, New York, NY, USA, 1–6.

[39] Peter Spindler, Ulf Schlichtmann, and Frank M Johannes. 2008. Abacus: Fast Legalization of Standard Cell Circuits with Minimal Movement. In *Proceedings of International Symposium on Physical Design*. Association for Computing Machinery, New York, NY, USA, 47–53.

[40] Chun-Ho Wu, Da-Zhi Wang, Andrew Ip, Ding-Wei Wang, Ching-Yuen Chan, and Hong-Feng Wang. 2009. A Particle Swarm Optimization Approach for Components Placement Inspection on Printed Circuit Boards. *Journal of Intelligent Manufacturing* 20 (2009), 535–549.

[41] Ling Zhang, Jack Juang, Zurab Kiguradze, Bo Pu, Shuai Jin, Songping Wu, Zhiping Yang, Jun Fan, and Chulsoon Hwang. 2022. Fast Impedance Prediction for Power Distribution Network Using Deep Learning. *International Journal of Numerical Modelling: Electronic Networks, Devices and Fields* 35, 2 (2022), e2956.

[42] Niansong Zhang, Xiang Chen, and Nachiket Kapre. 2022. Rapidlayout: Fast Hard Block Placement of FPGA-Optimized Systolic Arrays Using Evolutionary Algorithm. *Transactions on Reconfigurable Technology and Systems* 15, 4 (2022), 1–23.