ModuPlace: LLM-Assisted Modular PCB Placement via Preference-Optimized Constraint Graph Generation

Yaohui Han[1], Beichen Li[2], Mingyang Zhao[3], Rongliang Fu[1,*], Qunsong Ye[4], Tinghuan Chen[2,*], Bei Yu[1], Tsung-Yi Ho[1]
[1]CUHK [2]CUHK-Shenzhen [3]Central South University [4]Index

Abstract

Despite the existence of various automated PCB placement frameworks, the industry still relies heavily on human engineers. This is because these frameworks cannot perform placement according to specific user requirements and preferences, which limits their flexibility and industrial adoption. Moreover, the existing frameworks lack a modular perspective in placement, resulting in outputs that often perform poorly and fail to meet practical requirements. To address these problems, we propose ModuPlace, leveraging the powerful capabilities of LLMs to perform modular PCB placement with preference-optimized constraint graph generation. In ModuPlace, the entire component set is partitioned into modules at different granularities, enabling hierarchical and modular placement. Moreover, ModuPlace utilizes fine-tuning and preference optimization to enhance the quality of constraint graph generation and align the results with those of human experts. Experimental results demonstrate that ModuPlace outperforms all baselines, achieving superior placement quality across all metrics.

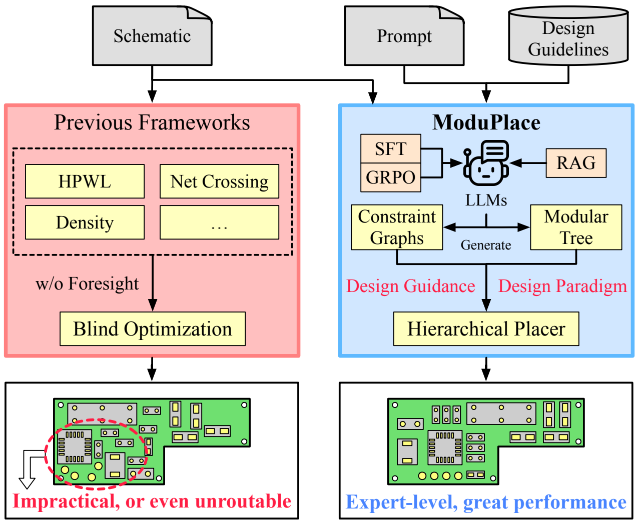

Figure 1: ModuPlace compared with previous frameworks for PCB placement. Previous frameworks typically select indirect metrics and blindly optimize without considering the practicality. ModuPlace utilizes LLMs to perform modular partition and generate constraint graphs to enhance the performance and make the placements close to the expert level.

1 Introduction

Printed circuit board (PCB) plays a critical role in nearly all industrial applications. As electronic products have become more versatile and complex, the number of nets and pins on modern PCBs is significantly increasing. However, designing these complicated PCBs presents significant challenges, which involve considering power and signal integrity [1–3], as well as many specific physical design rules and user requirements. Now, industrial PCBs still rely heavily on manual design by human experts.

The challenges of automated PCB design primarily stem from the complexity of placement. Specifically, each component can be rotated and placed on both sides of the board. The shape of the board frame and the components are often irregular. Therefore, the solution space of placement is extremely large in most circumstances. More importantly, in the industry, most PCBs are designed by engineers using a modular perspective to provide foresight for the subsequent routing. The modular design paradigm can save wirelength and reduce the number of vias, improving overall performance and practicality, but it usually requires expertise from human engineers.

To address the problems from PCB placement, various frameworks have been developed, typically integrating multiple design constraints with optimization objectives. OpenROAD [4] has open-sourced a simple yet effective PCB placer, SA-PCB [5], which utilizes simulated annealing to find a suitable placement solution. NS-Place [6] minimized net congestion using a support vector machine-like formulation and performed legalization by solving a routability-aware Mixed-Integer Linear Programming (MILP). HiePlace [7] proposed a heuristic algorithm to optimize the very high-density placement, thereby reducing manufacturing complexity. However, all these methods rely solely on indirect evaluation metrics (e.g., HPWL and density) to evaluate placement, so their results are often far from actual applications and lack foresight for routing.

Some studies have applied AI models to achieve expert-level PCB placements. DeepPCB [8] is a commercial tool that implements PCB PnR using reinforcement learning, aiming to reduce wirelength and via count in industrial PCBs. However, DeepPCB [8] still relies solely on traditional metrics (e.g., HPWL), and its overall performance and usability remain limited. PCBAgent [9], a framework that combines LLM and RL agents, is proposed to enhance PCB placement by considering constraints from multiple masks [10]. When the connections between components are simple, PCBAgent [9] may significantly reduce the solution space and be effective. Although this method may achieve low HPWL, its placement may be unroutable and cannot be utilized in actual applications. Meanwhile, it consumes high training costs and is extremely difficult to converge.

*Corresponding authors.
This work is supported by the National Key Research and Development Program of China (No. 2023YFB4402900), the National Natural Science Foundation of China (No. 62304197, 92573108), and Shenzhen Index Tech. This paper was jointly supported by the Research Grants Council of Hong Kong SAR No. CUHK14211324 and No. T46-415/25-R, and conducted in the JC STEM Lab of Intelligent Design Automation funded by The Hong Kong Jockey Club Charities Trust.

The success of LLMs has brought new opportunities to achieve high-quality automatic PCB placement. With proper supervised fine-tuning (SFT) and preference optimization, LLM can better understand design guidelines and user requirements for PCB placement, and learn from the unquantifiable preferences of human engineers. More importantly, LLM can optimize key performance indicators, such as electromagnetic (EM) performance, by combining expertise from experts. This paper proposes ModuPlace, an LLM-assisted modular PCB placement framework via preference-optimized constraint graph generation. Compared to previous frameworks, ModuPlace considers both modular information and design guidance from generated constraint graphs to achieve better PCB placement results, as shown in Figure 1.

Overall, our contributions are as follows:

(1) To the best of our knowledge, this paper proposes the first modular-designed PCB placement framework, which matches that of manual design by human experts.
(2) We propose the comprehensive constraint graph (CCG) to enable LLMs to generate this format, guiding hierarchical placement, alleviating hallucinations, and improving overall performance.
(3) To bridge the gap between LLM agents and human engineers in PCB placement, ModuPlace utilizes a two-stage specialization method, with the implementation of SFT and GRPO for CCG generation enhancement.
(4) Experimental results demonstrate that our ModuPlace can significantly save wirelength and vias, and improve the EM performance compared with all baselines.

2 Preliminary

2.1 Modular PCB Placement

In actual applications, most expert-level PCB placements are designed in a modular perspective. The advantages of this method include reduced wirelength and via count, minimized EM interference between components, and decreased manufacturing complexity. Modular PCB placement involves dividing the entire component set into multiple functional modules, and each module can be treated as a cohesive unit during the design process.

None of the existing automated PCB placement frameworks employs a modular perspective. This results in the components being placed without strategic organization, often leading to poor performance and limited practical usability. This is one of the main reasons why none of the existing frameworks has been widely adopted in industry.

2.2 Group Relative Policy Optimization

There are several methods for training the LLM agent to align to human preferences, i.e., Reinforcement Learning from Human Feedback (RLHF) [11, 12]. Among them, the most influential method is Proximal Policy Optimization (PPO) [13]. However, PPO requires training a value network as a critic to precisely estimate the reward function. In some complicated tasks, such as PCB placement, achieving this is exceptionally challenging.

Group Relative Policy Optimization (GRPO) [14] is an effective and straightforward method to perform RLHF [11, 12]. It is a policy gradient method that extends PPO by incorporating group-wise advantage normalization techniques to improve the parameter update process. Regarding the PCB placement task, assigning scores to the results is unstable and difficult. But determining their relative ranks is much easier. Therefore, implementing GRPO in PCB placement is a viable method that could enable agents to learn from human expertise and generate expert-level placements.

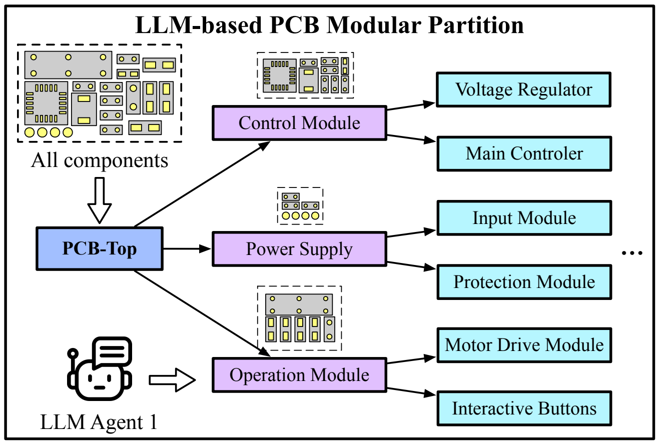

Figure 2: An example of the LLM-based PCB modular partition. The source node of HMT represents the complete set of all components. As the number of layers increases, the granularity of the partitioning becomes finer.

3 ModuPlace

3.1 Overall Flow

ModuPlace is an LLM-assisted framework for PCB placement that leverages a modular design perspective and combines the expertise and preferences of human experts. The overall flow of ModuPlace can be divided into the following three parts:

(1) With the guidance of the extracted graph-format netlist and design guidelines for RAG, an LLM agent is deployed to perform the modular partition and construct the hierarchical modular tree, detailed in Section 3.2.
(2) To generate high-quality CCGs, another LLM agent is fine-tuned using referenced PCBs. A customized GRPO is then deployed to perform further optimization, meeting the industrial requirements and expert preferences, detailed in Section 3.3.
(3) With the hierarchical modular tree and CCGs, ModuPlace iteratively performs hierarchical PCB placement and achieves expert-level placement results, detailed in Section 3.4.

3.2 LLM-based PCB Modular Partition

In modern PCB design, engineers typically employ a modular approach to decrease costs and enhance performance. To apply this perspective in ModuPlace, we propose the hierarchical modular tree (HMT) to strengthen the effectiveness of modular partition through an LLM agent. As shown in Figure 2, in HMT, the source node contains all components of the PCB, and the nodes closer to the bottom layer represent modules partitioned into smaller granularities. An LLM agent is employed to perform hierarchical partitioning, using the schematic as input and external design guidelines for RAG, to generate the proper HMT. Specifically, the agent is prompted to partition each module represented by parent nodes into finer granularities, as shown in Figure 2. Crucially, the agent incorporates an internal confidence assessment mechanism [15], which evaluates whether the current module requires further partitioning. The hierarchical modular partition is applied recursively until no leaf node has a confidence level exceeding the prescribed threshold. Additionally, similar to [16, 17], we construct a netlist heterogeneous graph (NHG) and perform sub-circuit matching [16] to generate a simplified format for the LLM, enabling it to better understand the netlist and build HMTs of higher quality and accuracy.

Through HMT, guidance for placement from the schematic, user queries, and external design documents can be well-integrated and utilized. The modular information represented in the HMT also enables subsequent hierarchical placement, enhancing design quality.

3.3 LLM-driven CCG Generation

Comprehensive Constraint Graph. For the complex PCB placement task, directly deploying LLMs to generate placement files is challenging and may lead to severe hallucination problems. Meanwhile, in practical PCB placement, the relative positions and rotations between different components require careful and detailed consideration. Therefore, we propose a new constraint graph type, the comprehensive constraint graph (CCG), to provide effective guidance for LLM-assisted hierarchical PCB placement.

The conventional mixed constraint graph (MCG) [18] only captures approximate relations between components or modules, which are insufficiently precise for placement. Moreover, in the MCG [18], positional constraints are limited to pairs of adjacent components or modules, severely limiting their effectiveness in PCB placement. The proposed CCG adopts a graph format $\mathcal{G}^c = (\mathcal{V}^c, \mathcal{E}^c, \mathcal{W}^c)$. Compared to MCG [18], our CCG provides more specific edge placement information. Specifically, the edge $\mathcal{E}^c_{ab}$ includes relative position $(\hat{x_{ab}}, \hat{y_{ab}})$ and rotation offsets $\hat{\theta_{ab}}$ for the component or module pair (v^c_a, v^c_b), and weight w^c_{ab} for the constraint. Among them, the relative position offset term includes both the absolute horizontal and vertical position relationships between components or modules, and the rotation offset term defines the ideal relative angle between them. Meanwhile, the edge weights in CCG quantitatively reflected the importance of these position and rotation constraints. These physical constraints, represented in CCG, can significantly support the subsequent hierarchical placement process.

In ModuPlace, the CCG generation is a crucial process for determining the quality of the placement results. However, without specialized optimization, the pre-trained LLM agent lacks the ability to generate high-quality CCGs. Meanwhile, the actual quality of placement is difficult to evaluate accurately using conventional indicators (e.g., HPWL and density). Human experts usually have foresight for the subsequent routing process, which can not only save wirelength and via but also ensure overall performance and usability. Therefore, aligning human experts' preferences during the CCG generation process is vital.

Two-stage Specialization. For stability and performance considerations, we propose a two-stage specialization method for post-training of LLM to enhance the quality of generated CCGs, corresponding to the SFT and GRPO stages, as shown in Figure 3.

In the SFT stage, we extract CCGs as ground truths from expert-designed PCB placements, thereby training the LLM to increase the probability of generating high-quality CCGs. The training is performed with the following customized loss function, for CCG g_j:

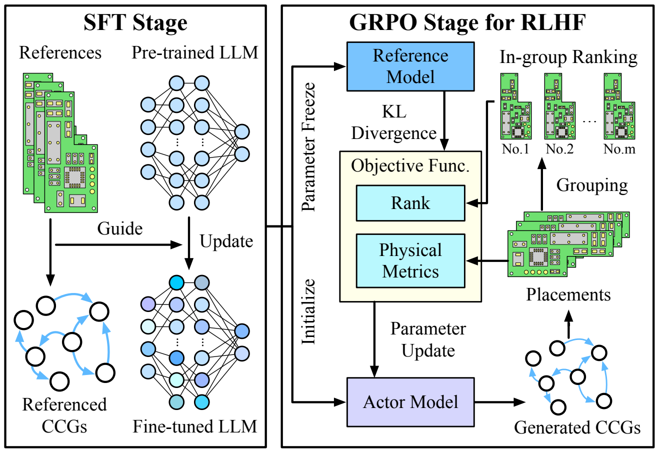

Figure 3: The workflow of two-stage specialization, including SFT and GRPO for the LLM to enhance the quality of CCGs.

$$\mathcal{L}_j = \lambda_1 \cdot \mathrm{HPWL}_j + \lambda_2 \cdot \mathcal{N}_j - \lambda_3 \cdot \mathrm{sim}(g_j, g_{\mathrm{ref}}), \tag{1}$$

where the $\lambda_{1,2,3}$ is the hyper-parameter set to balance the three terms. The first two terms represent the physical metrics, including HPWL and net crossing $\mathcal{N}$. The last term represents the similarity between the embeddings of the generated CCG g_j and the referenced CCG g_{ref} through GraphSAGE [19]. This stage provides an explicit CCG generation benchmark for the LLM agent.

The subsequent GRPO stage is based on the fine-tuned LLM to perform RLHF. Specifically, the LLM agent generates multiple CCGs under the same queries. These CCGs are then organized into groups. Manual scoring of placements may lead to instability, including inconsistent standards and difficulty in quantification. Therefore, in the GRPO stage, a trained reward model utilizes rankings within these groups to represent relative advantages, enabling GRPO to be performed under the advantage $\hat{A}$, as shown in Equation (5). The deployment details will be introduced in the following subsection.

In the two-stage specialization method, ModuPlace can first be fine-tuned on expert-level CCG generation benchmarks, and then gradually improve the generation probability of high-quality CCGs by using GRPO. This method bridges the gap between automated placement frameworks and human engineers, thereby robustly enhancing the quality of CCG generation.

Customized Adaptation of GRPO. Specifically, ModuPlace samples a group of queries and output CCGs $\mathcal{G}^c_i = \{g^c_1, g^c_2, ..., g^c_m\}$ from the old policy $\pi_{\theta_{\mathrm{old}}}$, then optimizes the policy model by maximizing the following customized objective function:

$$\mathcal{J}(\theta) = \mathbb{E}[q \sim P(Q), \mathcal{G}^c_i \sim \pi_{\theta_{\mathrm{old}}}(O|q)]$$
$$\frac{1}{n}\sum_{i=1}^{n}\frac{1}{m}\sum_{j=1}^{m}\left[\min\left(I_j\hat{A}_j, \mathrm{clip}(I_j, 1-\epsilon, 1+\epsilon)\hat{A}_j\right)\right] - \beta\mathbb{D}_{\mathrm{KL}_j}, \tag{2}$$

where I_j represents the probability ratio of the current policy π_θ to the old policy $\pi_{\theta_{\mathrm{old}}}$ in generating a specific CCG g^c_j, as follows:

$$I_j = \frac{\pi_\theta(g^c_j|q)}{\pi_{\theta_{\mathrm{old}}}(g^c_j|q)}, \tag{3}$$

and clip(.) is a function to restrict to a specified range in a parameter update round. ϵ is the hyper-parameter to set the min-max bound of

Algorithm 1 CCG Equivalence Determination

```
Input: two CCG $g_a^c$ and $g_b^c$ sharing the same node set $\mathcal{V}^c$, common
    edge count threshold $\tau$, cosine similarity threshold $\psi$.
Output: a boolean value indicating whether the two CCGs can be
    considered equivalent in the customized GRPO.
1: cnt $\leftarrow 0, \boldsymbol{\rho} \leftarrow \emptyset, \boldsymbol{w_a} \leftarrow \emptyset, \boldsymbol{w_b} \leftarrow \emptyset$
2: for each node pair $(u, v) \in \mathcal{V}^c \times \mathcal{V}^c$ do
3:     if $(u, v) \in \mathcal{E}_a^c$ and $(u, v) \in \mathcal{E}_b^c$ then
4:         cnt $\leftarrow$ cnt + 1, $\boldsymbol{\rho} \leftarrow \boldsymbol{\rho} \cup \{(u, v)\}$
5: $\boldsymbol{\rho} \leftarrow$ Sort($\boldsymbol{\rho}$)
6: for each edge $e \in \boldsymbol{\rho}$ do
7:     $\boldsymbol{w}_a \leftarrow \boldsymbol{w}_a \cup \{w_a^c(e)\}, \boldsymbol{w}_b \leftarrow \boldsymbol{w}_b \cup \{w_b^c(e)\}$
8: if cnt $\geq \tau$ and $\frac{\boldsymbol{w}_A \cdot \boldsymbol{w}_B}{\|\boldsymbol{w}_A\| \cdot \|\boldsymbol{w}_B\|} \geq \psi$ then
9:     return True
```

the clip(.). $\mathbb{D}_{\mathrm{KL}_j}$ is the Kullback-Leibler divergence, which integrates the unbiased estimator [20] to guarantee its positive value, as shown in Equation (4). This term controls the constraint from the reference policy π_{ref} and prevents the updated π_θ from deviating too much.

$$\mathbb{D}_{\mathrm{KL}_j} = \frac{\pi_{\mathrm{ref}}(g_j^c|q)}{\pi_\theta(g_j^c|q)} - \log \frac{\pi_{\mathrm{ref}}(g_j^c|q)}{\pi_\theta(g_j^c|q)} - 1, \tag{4}$$

The advantage is calculated based on both the actual physical metrics and the ranking $\hat{r}_j$ of the output CCG g_j inside each group. Among them, the in-group rankings are provided by a graph neural network, trained on ranking data from human expert feedback. Its inputs are two distinct CCGs, and the output is the ranking. In this way, relative advantage within groups of any size can be determined. This method turns the conventional reward into multiple pairwise preferences, ensuring the stability of GRPO. Different from the conventional settings in GRPO [14], the advantage $\hat{A}_j$ of our customized GRPO is that:

$$\hat{A}_j = \omega_1 \cdot \frac{\mu_{\mathrm{HPWL}} - \mathrm{HPWL}_j}{\sigma_{\mathrm{HPWL}}} + \omega_2 \cdot \frac{\mu_{\mathcal{N}} - \mathcal{N}_j}{\sigma_{\mathcal{N}}} + \omega_3 \cdot e^{-\hat{r}_j}, \tag{5}$$

where $\omega_{1,2,3}$ are hyperparameters to balance the impact of the three terms. μ_{HPWL} and $\mu_{\mathcal{N}}$ are the average of HPWL and net crossing within the group, respectively. σ represents the standard deviation.

CCG Equivalence Determination. Through the above settings, we have incorporated human expert preferences, which are difficult to quantify, into the feedback reward of the customized GRPO. However, the CCGs may be relatively large, so in actual circumstances, complete CCG consistency is extremely rare, and it is hard to calculate I. Therefore, we propose a CCG equivalence determination method to redefine the equivalence between CCGs in our customized GRPO, enabling I to be calculated correctly and efficiently.

As shown in Algorithm 1, the CCG equivalence determination primarily involves two aspects: common edges count and cosine similarity. A common edge refers to an edge in two CCGs that links the same nodes, meaning they correspond to the same component pair (lines 2-4). Then, the common edge set will be sorted in the same order (line 5), and the weights of these common edges, representing the positional and rotational constraints, will be stored in the form of vectors (lines 6-7). Only if the common edge count and cosine similarity of the two CCGs exceed the preset threshold will they be treated as equivalent in the customized GRPO (lines 8-9).

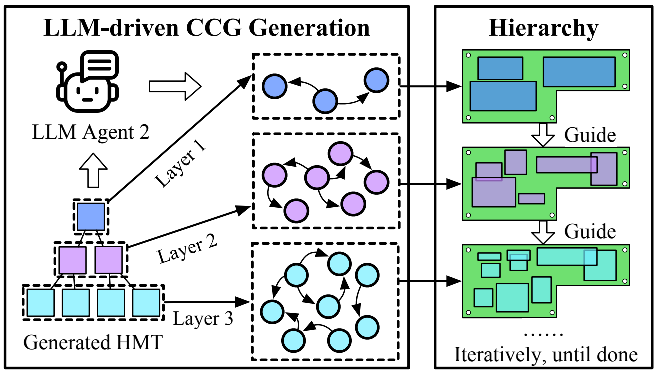

Figure 4: In the process of LLM-driven CCG generation, an LLM agent takes the modular information from HMT as input, and generates CCGs layer by layer to support the subsequent hierarchical placement.

3.4 Hierarchical PCB Placement

Each layer of HMT corresponds to a CCG, which is generated from different modular information and descriptions. Since the components have been organized into modules in the HMT, ModuPlace then performs hierarchical placement guided by the generated CCGs, as shown in Figure 4.

The first issue is to place the modules in a hierarchical structure layer by layer. The size of each module is determined by its corresponding components and then refined through optimization of HPWL. Components are not allowed to overlap in the area of the corresponding module. In this method, the sizes of the modules can be estimated and determined. The formulation is shown as follows:

$$\min\ (t_1 \cdot \mathrm{BoundingBox}(C_q) + t_2 \cdot \sum_{k=1}^{|net|} \mathrm{HPWL}(x, y, \theta)),$$
$$\text{s.t. } \theta_i \in \{0^\circ, 90^\circ, 180^\circ, 270^\circ\},\ \forall i \in C_q, \tag{6}$$

where C_q means the set of components in module q, $t_{1,2}$ is the hyperparameter set to balance the impact of the two terms. BoundingBox(.) represents the area of the bounding box that contains the whole current component set. After optimization, the bounding box represents the area of the module.

As shown in Figure 5, all the areas of placed modules act as guidance for the placement of sub-modules in the next layer of HMT. The formulation of each step in the CCG-based hierarchical PCB placement is as follows:

$$\min \sum_{i=1}^{|S_q|} \sum_{j=i}^{|S_q|} w_{ij}^c \cdot [\gamma_1 \cdot (|\hat{x_{ij}} - x_{ij}| + |\hat{y_{ij}} - y_{ij}|)$$
$$+ \gamma_2 \cdot |\hat{\theta_{ij}} - \theta_{ij}|] + \gamma_3 \cdot A_i^q + \gamma_4 \cdot \sum_{k=1}^{|net|} \mathrm{HPWL}(x, y, \theta),$$
$$\text{s.t. } \theta_i \in \{0^\circ, 90^\circ, 180^\circ, 270^\circ\},\ \forall i \in S_q, \tag{7}$$

where S_q is the set of sub-modules in module q, w_{ij}^c is the weight on the edge of sub-module i and j in CCG, $(\hat{x_{ij}}, \hat{y_{ij}})$ and $\hat{\theta_{ij}}$ define the position and rotation offsets in CCG between sub-modules, respectively, A_i^q is the common area between module q and sub-module i. $\gamma_{1,2,3,4}$ is the hyper-parameter set to balance the multiple objectives. After the modules in the penultimate layer of HMT are well-placed, S_q

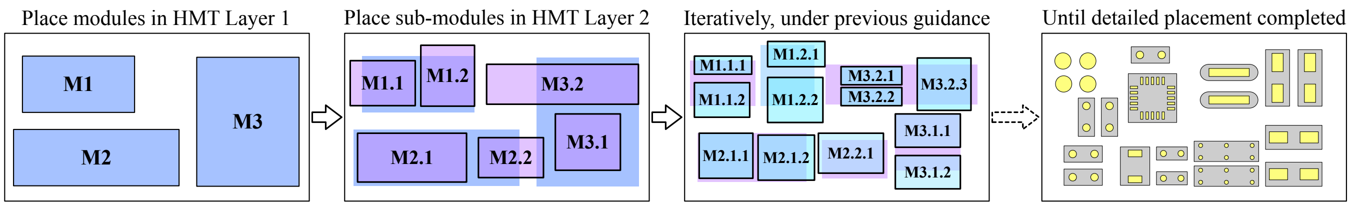
Place modules in HMT Layer 1
M1
M2
M3
Place sub-modules in HMT Layer 2
M1.1
M1.2
M3.2
M2.1
M2.2
M3.1
Iteratively, under previous guidance
M1.1.1
M1.1.2
M1.2.1
M1.2.2
M3.2.1
M3.2.2
M3.2.3
M3.1.1
M3.1.2
M2.1.1
M2.1.2
M2.2.1
Until detailed placement completed

Figure 5: An example of hierarchical PCB placement. ModuPlace first performs placement for modules M1-M3. Each module can be divided into sub-modules with finer granularity, such as M1.1. And then ModuPlace places these sub-modules under the guidance of the last placement result. ModuPlace will repeat this process until all sub-modules are placed and complete the detailed placement for all components.

represents the set of components in module q, ModuPlace performs the detailed placement for components, and no more overlapping is allowed, as shown in Figure 5.

4 Experiment Results

4.1 Experiment Setting

We utilize ModuPlace in Python, PyTorch, and HuggingFace Transformers for LLM inference and specialization, as well as C++ for the implementation of detailed PCB placement. The Boost C++ Library [21] is used to perform geometric computations within ModuPlace. We perform SFT and GRPO based on Qwen-2.5 (7B) for ModuPlace, and run the experiments on a Linux server equipped with 76 Intel Xeon CPU cores and 4 NVIDIA A100 GPUs, each with 80GB memory.

Regarding the specialization stage, we utilize 426 CCGs corresponding to expert-level placements as training data. In the GRPO stage, the LLM agent generates 36 CCGs for each input, and each group for GRPO consists of 4 CCGs. To verify the performance of PCB placements from a practical perspective and ensure they are routable, we perform routing on all placement results, demonstrating end-to-end performance, rather than relying on conventional indirect metrics. We implement PcbRouter [22] and Freerouting [23] to achieve the final layout designs. To avoid bias, we rerouted the placement results multiple times under different configurations and then averaged the results. EM performance is a pretty critical indicator for measuring whether a PCB can be used in industrial applications [1, 3]. Thus, we use Keysight ADS [24] to simulate the key EM indicators of PCB layouts: insertion loss (IL) and crosstalk (XT) [2]. Lower IL and XT indicate better EM performance. These two metrics are both expressed in decibels (dB) in our experiments.

4.2 Benchmarks and Baselines

All PCB layout benchmarks are from OpenROAD [4] and real-world industrial applications, which are shown in Table 1. The baselines in our experiments are as follows:

(1) **SA-PCB** [5]: an open-sourced and effective PCB placer, released by OpenROAD [4]. And the placement process is implemented using a simulated annealing algorithm.
(2) **NS-Place**[6]: a popular framework from academia for PCB placement that reduces net congestion to enhance routability.
(3) **DeepPCB** [8]: a powerful commercial tool for PCB PnR based on reinforcement learning; its placer leverages lots of design experience through training.

Table 1: Benchmark statistics.

Case	#Components	#Fixed	#Pins	#Nets	Boundary (mm)	#Layers
P1	62	4	231	81	55 × 55	2
P2	49	1	161	54	41 × 40	2
P3	35	1	138	38	55 × 28	2
P4	64	3	312	63	86 × 71.5	4
P5	60	2	233	35	58 × 59.5	4
P6	46	3	113	40	102 × 60	2
P7	72	4	167	52	64 × 55	2
P8	55	6	129	43	100 × 55	2
P9	48	2	105	21	97 × 15.5	2
P10	97	4	214	61	98 × 55	2
P11	71	6	176	57	90 × 59	2

(4) **PCBAgent** [9]: an effective LLM-based framework for placement, utilizes an RL agent to perform optimization, and multiple masks [10] to generate design constraints.

4.3 Overall Performance

The overall results of PCB placements after routing on all benchmarks are shown in Table 2. Compared with SA-PCB [5] and NS-Place [6], our ModuPlace achieves average reductions of 17.14% and 15.01% in wirelength, and 40.51% and 32.02% in via count, respectively. Even compared with the commercial tool DeepPCB [8], ModuPlace still generates superior placements, reflected in significantly lower average wirelength and via counts after routing.

PCBAgent [9] utilizes multiple masks [10] to increase surface layer wires (SLW), aiming to improve routing quality and reduce HPWL. While a higher SLW may reduce HPWL, it often fails to yield a smaller wirelength and via count after routing. With larger component counts or more complex netlists, this method may cause components within the same net to form multiple localized clusters, creating routing blockages and significantly decreasing routability. Therefore, experimental results show that PCBAgent [9] performs better on relatively small-scale PCB designs, such as P6 and P8. Compared to PCBAgent [9], ModuPlace achieves a 14.26% reduction in wirelength and a 48.51% reduction in via count after routing.

DeepPCB [8] performs relatively well in terms of wirelength and via count due to its implementation of extensive historical designs to train the reward model. As shown in Table 2, the average number of vias is close to that of ModuPlace, and the wirelength is 17.22% higher than ModuPlace. With the help of an RL agent, it indeed possesses some foresight in finding potentially better placements, which save wirelength and vias after routing. However, due to the

Table 2: Performance comparison of PCB placements after routing.

Case	SA-PCB [5]				NS-Place[6]				DeepPCB[8]				PCBAgent [9]				**ModuPlace (Ours)**			
	WL	Via	EM Perf.		WL	Via	EM Perf.		WL	Via	EM Perf.		WL	Via	EM Perf.		WL	Via	EM Perf.	
			↓ IL	↓ XT			↓ IL	↓ XT			↓ IL	↓ XT			↓ IL	↓ XT			↓ IL	↓ XT
P1	1762.9	161	7.75	-34.86	1516.2	66	7.65	-35.09	1195.5	51	9.81	-39.02	1679.8	83	8.10	-34.81	1206.8	47	5.49	-40.79
P2	1209.9	77	2.87	-40.44	1028.4	68	3.86	-38.97	991.5	45	5.89	-42.66	1206.1	90	3.55	-39.65	978.1	40	2.35	-45.05
P3	768.5	56	2.43	-36.55	549.0	27	5.72	-36.20	736.8	39	6.31	-36.57	693.8	33	3.87	-38.23	587.4	30	2.28	-41.17
P4	4390.6	215	6.18	-41.25	3361.2	147	6.90	-30.96	3700.5	106	6.96	-38.48	4175.3	249	5.82	-41.06	3077.4	129	4.55	-44.71
P5	1708.1	51	8.73	-40.59	1872.4	78	9.15	-30.77	1527.5	53	10.04	-40.65	1810.1	92	8.32	-39.17	1594.7	47	8.06	-43.28
P6	649.4	40	11.28	-36.96	1027.9	44	7.54	-37.68	685.4	30	7.43	-38.23	419.7	43	8.78	-33.37	483.6	39	6.87	-37.91
P7	1489.4	89	8.63	-41.91	1518.4	70	11.67	-42.25	1393.6	39	9.26	-42.50	1214.3	97	8.79	-40.29	1192.7	56	8.00	-38.80
P8	688.6	12	8.56	-38.07	1129.3	89	8.68	-37.99	917.8	26	10.38	-39.04	586.7	40	9.35	-33.75	816.2	23	8.62	-40.88
P9	741.1	20	2.33	-39.51	909.8	39	2.44	-32.05	990.3	38	2.21	-40.72	825.2	43	1.98	-35.93	728.6	29	2.56	-42.10
P10	2165.1	144	3.92	-45.01	2099.6	114	3.58	-40.67	1946.8	54	7.43	-46.31	2056.7	201	4.02	-45.00	1883.5	78	2.84	-43.29
P11	1253.8	34	9.06	-36.32	1379.8	45	8.61	-36.14	1345.6	56	10.20	-35.79	1579.4	67	10.30	-39.84	1382.3	21	7.79	-42.16
Avg.	1528.5	81.7	6.52	-39.22	1490.2	71.5	6.89	-36.25	1484.6	48.8	7.81	-39.99	1477.3	94.4	6.62	-38.28	**1266.5**	**48.6**	**5.40**	**-41.83**

Table 3: Comparison between LLM implementations.

Model	Performance			
	WL	Via	↓ IL	↓ XT
ModuPlace (7B)	**1266.5**	**48.6**	**5.40**	**-41.83**
Qwen-2.5 (7B)	1928.7	97.3	8.82	-35.6
Qwen-3 (14B)	1866.1	99.3	8.26	-35.87
InternLM-2.5 (20B)	1874.6	96.7	8.69	-36.12
Deepseek-R1 (32B)	1824.8	94.6	8.33	-36.91
Claude-3.5-Sonnet (API)	1707.5	92.1	7.26	-36.2
GPT-4o (API)	1756.8	84.6	7.85	-37.19

lack of guidance from human expertise and preference, the EM performance is significantly low, indicating its practicality is relatively poor compared with our ModuPlace.

As for EM performance, ModuPlace demonstrates significantly superior IL and XT scores across almost all benchmarks compared to all baselines. As shown in Table 2, compared to the four baselines, ModuPlace achieves improvements of 17.17%, 21.62%, 30.85%, and 18.42% in IL, and 6.65%, 15.39%, 4.6%, and 9.27% in XT, respectively. ModuPlace achieves such excellent EM performance for two primary reasons: the implementation of SFT and GRPO, as well as the modular design approach. EM performance is consistently considered in expert-manual placement designs; therefore, applying SFT and GRPO enables the LLM to learn from historical designs and enhance the quality of placement. Additionally, because components belonging to the same module typically share specific nets and have similar operating frequencies and power requirements, modular guidance is also vital for improving EM performance.

4.4 Evaluation of LLM Implementations

To further verify the effectiveness of the optimization for CCG generation in ModuPlace, we compared the results of ModuPlace with those of different pre-trained LLMs [25–30], as shown in Table 3. We let all these LLMs generate CCGs and perform hierarchical placement. Notably, all locally deployed LLMs [25–28] exhibited significantly poor performance compared with our ModuPlace. Even for LLMs with larger numbers of parameters, GPT-4o [30] and Claude-3.5-Sonnet [29] still show substantially worse performance compared to ModuPlace. Specifically, even for the best results achieved by these two LLMs, ModuPlace still shows 25.82% less wirelength, 42.56% fewer vias, and 25.61% and 15.55% better performance in IL and XT, respectively. Results in Table 3 directly demonstrate the effectiveness and performance of ModuPlace.

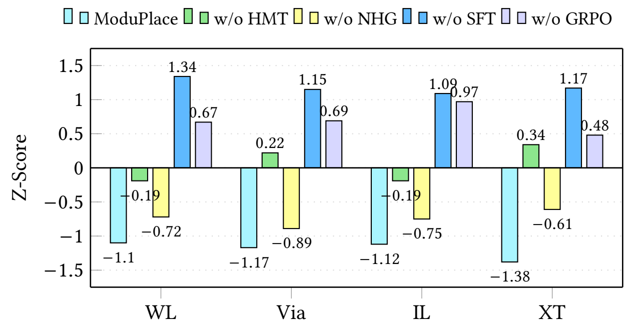

Figure 6: Comparison between different configurations of ModuPlace. The results are processed using z-score normalization, and higher scores indicate a greater importance of the removed part in improving the performance of ModuPlace.

4.5 Ablation Study

To demonstrate the effectiveness of each part in ModuPlace, we conducted an ablation study for different configurations, as shown in Figure 6. Overall, without any of these parts in ModuPlace, it fails to perform optimally, demonstrating their importance for PCB placement. It is notable that ModuPlace without HMT shows significantly worse performance across all four metrics. This directly proves the effectiveness of the hierarchical modular placement design method. Additionally, SFT and GRPO greatly help improve the overall performance of ModuPlace. Specifically, there is 35.1% more wirelength and 66.2% more vias, 39.2% more IL, and 15.5% more XT for ModuPlace without SFT. Also, there is 25.4% more wirelength and 53.1% more vias, 41.5% more IL, and 11.3% more XT for ModuPlace without GRPO.

5 Conclusion

This paper proposed ModuPlace, an LLM-assisted modular PCB placement framework. ModuPlace is the first automated framework that performs PCB placement using a modular perspective, similar to that of human engineers. We also proposed the two-stage specialization, including SFT and GRPO, to bridge the gap between LLM agents and human engineers, achieving superior placement results. Experimental results on industrial PCB benchmarks demonstrate that ModuPlace outperforms existing state-of-the-art baselines across all metrics, making it more suitable and flexible for practical industrial applications.

References

[1] T.-L. Wu, H.-H. Chuang, and T.-K. Wang, "Overview of power integrity solutions on package and PCB: Decoupling and EBG isolation," *IEEE Transactions on electromagnetic compatibility*, vol. 52, no. 2, pp. 346–356, 2010.

[2] E. Bogatin, *Signal and power integrity–simplified*. Pearson Education, 2010.

[3] A. Mircea, "Main aspects concerning PCB manufacturing optimization," *Circuit World*, vol. 38, no. 2, pp. 75–82, 2012.

[4] OpenROAD. [Online]. Available: https://github.com/the-openroad-project

[5] SA-PCB. [Online]. Available: https://github.com/The-OpenROAD-Project/SA-PCB

[6] C.-K. Cheng, C.-T. Ho, and C. Holtz, "Net Separation-Oriented Printed Circuit Board Placement via Margin Maximization," in *Proc. ASPDAC*, 2022, pp. 288–293.

[7] S. Li, Z. Zhuang, M. Liu, W. Sheng, B. Yu, and T.-Y. Ho, "HiePlace: Efficient hierarchical PCB placement," *IEEE TCAD*, 2025.

[8] DeepPCB. [Online]. Available: https://app.deeppcb.ai/

[9] L. Chen, R. Chen, S. Hu, X. Yao, Z. Tang, S. Kai, S. Xu, M. Yuan, J. Hao, B. Yu *et al.*, "PCBAgent: An Agent-based Framework for High-Density Printed Circuit Board Placement," in *Proc. ASPDAC*, 2025, pp. 781–787.

[10] Y. Lai, Y. Mu, and P. Luo, "Maskplace: Fast chip placement via reinforced visual representation learning," *Proc. NIPS*, vol. 35, pp. 24 019–24 030, 2022.

[11] P. F. Christiano, J. Leike, T. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," *Advances in neural information processing systems*, vol. 30, 2017.

[12] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving, "Fine-tuning language models from human preferences," *arXiv preprint arXiv:1909.08593*, 2019.

[13] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," 2017. [Online]. Available: https://arxiv.org/abs/1707.06347

[14] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu, and D. Guo, "DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models," 2024. [Online]. Available: https://arxiv.org/abs/2402.03300

[15] M. Xiong, Z. Hu, X. Lu, Y. Li, J. Fu, J. He, and B. Hooi, "Can LLMs Express Their Uncertainty? An Empirical Evaluation of Confidence Elicitation in LLMs," 2024. [Online]. Available: https://arxiv.org/abs/2306.13063

[16] Z. Wei, Z. Kong, Y. Wang, D. Z. Pan, and X. Tang, "TopoSizing: An LLM-aided Framework of Topology-based Understanding and Sizing for AMS Circuits," *arXiv preprint arXiv:2509.14169*, 2025.

[17] Y. Zhao, X. Jiang, Q. Xu, R. Wang, Y. Lin *et al.*, "DeepLayout: Learning Neural Representations of Circuit Placement Layout," in *Forty-second International Conference on Machine Learning*.

[18] X. Gao, H. Zhang, M. Liu, L. Shen, D. Z. Pan, Y. Lin, R. Wang, and R. Huang, "Interactive analog layout editing with instant placement and routing legalization," *IEEE TCAD*, vol. 42, no. 3, pp. 698–711, 2022.

[19] W. L. Hamilton, R. Ying, and J. Leskovec, "Inductive Representation Learning on Large Graphs," 2018. [Online]. Available: https://arxiv.org/abs/1706.02216

[20] J. Schulman. (2020) Approximating KL Divergence. [Online]. Available: http://joschu.net/blog/kl-approx.html

[21] Boost. [Online]. Available: https://www.boost.org/

[22] PcbRouter. [Online]. Available: https://github.com/The-OpenROAD-Project/PcbRouter

[23] Freerouting. [Online]. Available: https://www.freerouting.app/

[24] Keysight ADS. [Online]. Available: https://www.keysight.com/us/en/products/software/pathwave-design-software/pathwave-advanced-design-system.html

[25] Qwen, A. Yang, B. Yang, and et al., "Qwen2.5 Technical Report," 2025. [Online]. Available: https://arxiv.org/abs/2412.15115

[26] A. Yang, A. Li, B. Yang, and B. Z. et al., "Qwen3 Technical Report," 2025. [Online]. Available: https://arxiv.org/abs/2505.09388

[27] InternLM-2.5. [Online]. Available: https://huggingface.co/internlm/internlm2_5-20b-chat

[28] DeepSeek-AI, D. Guo, and D. Y. et al., "DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning," 2025. [Online]. Available: https://arxiv.org/abs/2501.12948

[29] Claude-3.5-Sonnet. [Online]. Available: https://www.anthropic.com/news/claude-3-5-sonnet

[30] OpenAI, A. Hurst, A. Lerer, and A. P. G. et al., "GPT-4o System Card," 2024. [Online]. Available: https://arxiv.org/abs/2410.21276