DevFormer: A Symmetric Transformer for Context-Aware Device Placement

Haeyeon Kim[*1] **Minsu Kim**[*2] **Federico Berto**[2] **Joungho Kim**[1] **Jinkyoo Park**[2]

Abstract

In this paper, we present DEVFORMER, a novel transformer-based architecture for addressing the complex and computationally demanding problem of hardware design optimization. Despite the demonstrated efficacy of transformers in domains including natural language processing and computer vision, their use in hardware design has been limited by the scarcity of offline data. Our approach addresses this limitation by introducing strong inductive biases such as relative positional embeddings and action-permutation symmetricity that effectively capture the hardware context and enable efficient design optimization with limited offline data. We apply DEVFORMER to the problem of decoupling capacitor placement and show that it outperforms state-of-the-art methods in both simulated and real hardware, leading to improved performances while reducing the number of components by more than 30%. Finally, we show that our approach achieves promising results in other offline contextual learning-based combinatorial optimization tasks.

1. Introduction

The development of artificial intelligence (AI) has been greatly facilitated by advancements in high-performance computing systems. However, as the need for faster data processing grows with recent advances in AI architecture scaling, the complexity of hardware design is increasing. As a result, human experts are no longer able to design hardware without the aid of electrical design automation (EDA) tools; despite their utility, the use of these tools is often hindered by long simulation times and insufficient computing resources, making the application of machine learning (ML) techniques in hardware design more and more crucial.

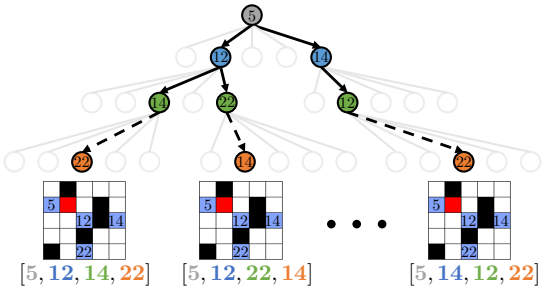


Figure 1: Example of action-permutation (AP) symmetric solutions: different action trajectories lead to the same solution group.

Recent studies have shown the potential of deep reinforcement learning (DRL) for sequential decision-making in various tasks in chip design, including chip placement (Mirhoseini et al., 2021; Agnesina et al., 2020), routing (Liao et al., 2019; 2020), circuit design (Zhao & Zhang, 2020), logic synthesis (Hosny et al., 2020; Haaswijk et al., 2018) and bi-level hardware optimization (Cheng & Yan, 2021). However, these methods have limitations. Firstly, they rely on online simulators, which are both slow and inaccurate. As a result, learning with offline expert data is more reliable, but such data is limited. Secondly, hardware design involves multi-level problems that change depending on the context, making it necessary to have a policy that can generalize to new problem contexts.

The Transformer has been recognized as a promising architecture for contextual models, owing to its capability to process sequential data in parallel, handle long-term dependencies and exhibit high expressivity (Vaswani et al., 2017). This has led to their widespread adoption in a variety of domains, including natural language processing (NLP) (Brown et al., 2020), computer vision (Dosovitskiy et al., 2020), graph representation learning (Ying et al., 2021), combinatorial optimization (Kool et al., 2019) and reinforcement learning (Chen et al., 2021). However, deriving an effective offline contextual policy with the Transformer requires a huge amount of offline data covering the broad context and input data space, which is often infeasible in the semiconductor industry.

[*]Equal contribution [1]Department of Electrical Engineering, Korea Advanced Institute of Science and Technology (KAIST) [2]Department of Industrial and Systems Engineering, Korea Advanced Institute of Science and Technology (KAIST). Correspondence to: Jinkyoo Park <jinkyoo.park@kaist.ac.kr>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

Training a flexible contextual design policy based on Transformer with a limited volume of offline data is a challenging task. The current study aims to overcome this issue by utilizing inductive biases for the target hardware design problem. For example, the device placement problem has an input order invariance property in that input permutations do not affect the output, while Transformers have a strong order bias. Fig. 1 illustrates this property: for example, $\{1, 2, 3, 4\}$ and $\{2, 3, 1, 4\}$ are considered action permutation (AP) symmetric solutions. In addition, the relative locations of hardware devices matter more than their absolute positions, while Transformer heavily uses an absolute positional encoding. Thus, we can exploit such properties of the target problem to modify Transformer architecture for deriving an effective contextual policy with a limited offline dataset.

In this paper, we present DEVFORMER, a novel transformer model for solving contextual offline hardware design problems. By incorporating strong domain-specific inductive biases, our model learns more efficiently and effectively, overcoming the limitations of traditional transformer architectures. We demonstrate the proposed approach on a novel hardware benchmark and validate its applicability on a real-world high bandwidth memory.

We summarize our contributions as follows:

- We propose a new positional embedding technique for the encoder of the transformer, utilizing relative chip location, and a probing-port context network (PCN) for the decoder. To address the issue of order bias in traditional positional embedding, we propose a recurrent context network (RCN) that only references the previously selected node. The PCN, RCN, and encoded node embeddings are used in an attentive manner to generate actions during the decoding process.

- To address order biases and impose AP symmetricity, we introduce a novel regularization loss term for the placement problem symmetricity. This approach can also be applied to other sequential design methods, such as DRL, with similar order bias issues.

- We demonstrate our approach to the decap placement problem (DPP) and release the novel DPP benchmark for researchers to evaluate and improve ML methodologies for hardware design challenges. As a means of promoting transparency and reproducibility, we make the source codes of our method and the baselines discussed in this paper publicly available online[1] as well as an accompanying interactive demonstration program[2] to facilitate engagement and experimentation.

[1] https://github.com/kaist-silab/devformer
[2] https://dppbench.streamlit.app

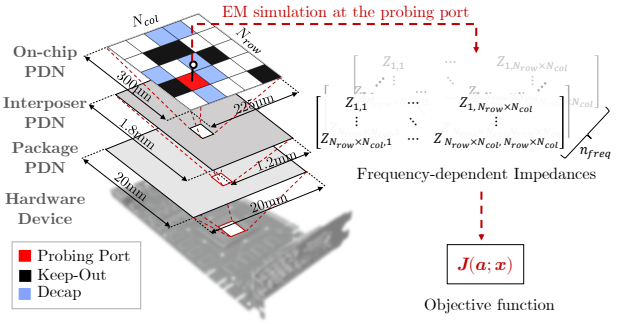


Figure 2: Grid representation of the target on-chip PDN of a hardware device and the EM simulation at the probing port to evaluate the objective function.

2. Preliminaries and Background

2.1. Decap Placement Problem (DPP)

This paper seeks to solve the decap placement problem (DPP), one of the essential hardware design problems. A decoupling capacitor (decap) is a hardware component that reduces power noise along the power distribution network (PDN) of hardware devices such as CPUs, GPUs, and AI accelerators, and improves the power integrity (PI).

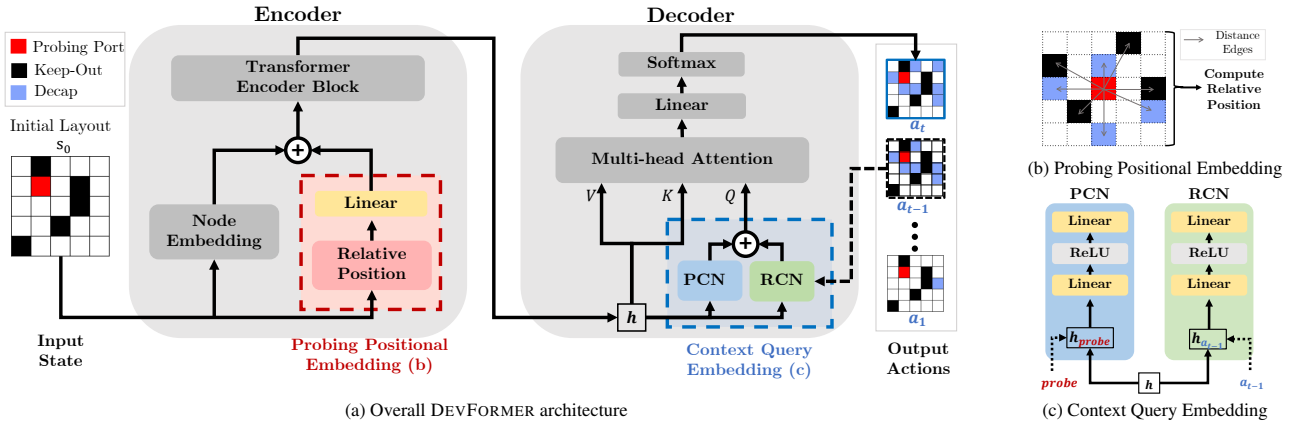
Formally, DPP is a black-box contextual optimization problem to find the optimal placement of decap $\boldsymbol{a} = \{\boldsymbol{a}_1, ..., \boldsymbol{a}_K\}$ that maximizes the PI objective $\mathcal{J}(;\boldsymbol{x})$. Note the objective is contextualized by the target hardware feature vectors $\boldsymbol{x}$ with the constraint of a limited number of decap K. Our research aims to solve:

$$\boldsymbol{a}^* = \arg\max_{a} \mathcal{J}(\boldsymbol{a}; \boldsymbol{x})$$
$$\text{s.t} \quad K \leq K^*, x \in \mathcal{X}$$

$\mathcal{X}$ refers to the context space of the target hardware. Note that the number of decap K is a crucial budget for DPP optimization as placing a decap is costly in semiconductor industries (Koo et al., 2018).

2.2. Contextual Markov Decision Processes (cMDP)

We formulate DPP as a contextual Markov decision process (Hallak et al., 2015, cMDP) to decompose the joint action policy into a sequential component policy to overcome the high dimensionality issue. Specifically, our objective function $\mathcal{J}(;\boldsymbol{x})$ is determined by the PDN, which is contextualized by $\boldsymbol{x}$. The contextualized PDN is represented as a set of three-dimensional feature vectors $\boldsymbol{x} = \{\boldsymbol{x}_i\}_{i=1}^{N_{\text{row}} \times N_{\text{col}}}$, where each grid (i.e., port) on the PDN is represented as $\boldsymbol{x}_i = (x_i, y_i, c_i)$, in which x_i, y_i indicate the 2D coordinates of the location; $c_i \in \{0, 1, 2\}$ indicates the condition of the port, whether it belongs to a probing port, keep-out regions, or decap allowed ports, respectively. See Appendix A.3 for further details.



(a) Overall DevFormer architecture

(b) Probing Positional Embedding

(c) Context Query Embedding

Figure 3: (a) Given design constraints and previous actions, the DevFormer model generates the optimal device placement action a_t. (b) Probing Positional Embedding (●) uses relative positions with respect to the probing port. (c) PCN (probing port context network ●) and RCN (recurrent context network ●) use embedding h to capture contextual information of initial design conditions and stages of the partial solution, respectively.

The design process sequentially places a pre-defined number K of decaps on the available PDN ports. We model this cMDP with state, action, and policy as follows.

State $\boldsymbol{s}_t$ contains the task-condition (i.e., problem context vectors) $\boldsymbol{x}$ and the previous selected actions: $\boldsymbol{s}_t = \{\boldsymbol{x}, \boldsymbol{a}_{1:t-1}\}$.

Action $a_t \in \{1, ..., N_{\text{row}} \times N_{\text{col}}\} \setminus \boldsymbol{s}_{t-1}$ is defined as an allocation of a decap to one of the available ports on PDN. The available ports are the ports on PDN, except for the probing port, keep-out, and previously selected ports. The concatenation of sequentially selected actions $\boldsymbol{a} = a_{1:K}$ indicates the final decap placement *solution*.

Policy $\pi_\theta(\boldsymbol{a}|\boldsymbol{x})$ is the probability of producing a specific solution $\boldsymbol{a} = a_{1:K}$, given the problem context vectors $\boldsymbol{x}$, and is factorized as:

$$\pi_\theta(\boldsymbol{a}|\boldsymbol{x}) = \prod_{t=1}^{K} p_\theta(a_t|\boldsymbol{s_t}), \tag{1}$$

where $p_\theta(a_t|\boldsymbol{s_t})$ is the segmented one-step action policy parameterized by the neural network.

The objective of DPP is to find the optimal parameter θ^* of the policy $\pi_\theta(\cdot|\boldsymbol{x})$ as:

$$\theta^* = \arg\max_\theta \mathbb{E}_{\boldsymbol{x}\sim p_\mathcal{X}(\boldsymbol{x})} \mathbb{E}_{\boldsymbol{a}\sim\pi_\theta(\cdot|\boldsymbol{x})} \left[\mathcal{J}(\boldsymbol{a};\boldsymbol{x})\right], \tag{2}$$

where $p_\mathcal{X}(\boldsymbol{x})$ is the probability distribution for varying *task-condition* $\boldsymbol{x}$ and $\mathcal{J}$ is objective function. Finding the optimal policy for various DPPs with changing conditions is a contextual learning problem, in which each DPP has a distinct context. Once the task $\boldsymbol{x}$ is sampled by $p_\mathcal{X}(\boldsymbol{x})$, the state-action space with complexity of $\binom{N_{\text{row}} \times N_{\text{col}}}{K}$ is determined. Then, an efficient policy $\pi_\theta(\boldsymbol{a}|\boldsymbol{x})$ should capture the contextual features among varying task conditions $\boldsymbol{x}$.

Note that DPP is an episodic task, where the reward is defined as the objective of the final state solution; the reward is the same as objective $\mathcal{J}$.

2.3. Objective Function $\mathcal{J}$

The objective function is a black-box simulator based on the electrical magnetic (EM) simulation or lab-level electrical measurement on fabricated products. To benchmark DPP as an ML task, we implemented the objective function with approximated modeling of electrical components as the fast Python simulator. As shown in Fig. 2, calculating objective function requires frequency-dependent impedances calculated through EM simulation at the probing port, and such process is costly in terms of time and computation. See Appendix A.4 for a detailed description of our modeling of the objective function.

3. Methodology

3.1. DevFormer Architecture

DevFormer, illustrated in Fig. 3, is a novel transformer-based architecture that incorporates hardware-aware prior components, such as the probing port positional embedding (●) for the encoder and the probing port context network (●) and recurrent context network (●) for the decoder. The encoder, f_{enc}, maps the task condition $\boldsymbol{x}$ to a high-dimensional embedding $\boldsymbol{h}$, through the function $f_{\text{enc}}(\boldsymbol{x})$. Similarly to the decoding scheme in pointer network (PointerNet) (Vinyals et al., 2015) and attention model (AM) (Kool et al., 2019), our decoder $p_{\text{dec}}(\boldsymbol{a}_t|\boldsymbol{h}, \boldsymbol{a}_{t-1})$ samples the indices of the placements in an auto-regressive manner: $\boldsymbol{a}_t \sim p_{\text{dec}}(\boldsymbol{a}_t|\boldsymbol{h}, \boldsymbol{a}_{t-1})$.

Encoder Our encoder consists of L layers of multi-head attention (MHA) and feedforward (FF), akin to the transformer network proposed in Vaswani et al. (2017) as illustrated in Fig. 3a. Before processing by MHA and FF, the task condition $\boldsymbol{x}$ is processed by two networks: node embedding and probing port positional embedding (PPE ●).

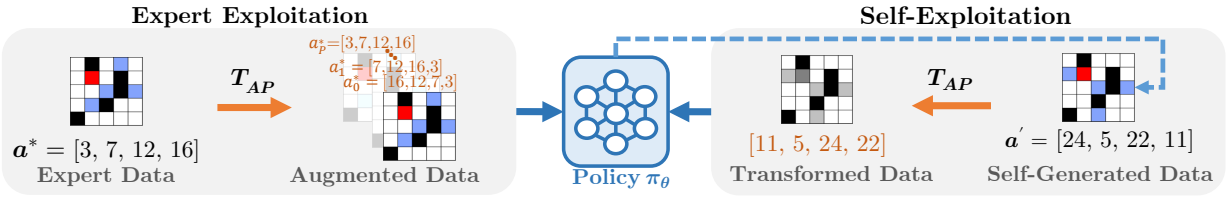


Figure 4: Collaborative symmetricity exploitation (CSE) framework with DEVFORMER policy. Training is carried out via expert exploitation (EE), which employs offline high-quality data, while self-exploitation (SE) uses data generated by a copy of the policy itself. AP-symmetricity is imposed through the T_{AP} transformation on both expert and self-generated data.

The node embedding is the node-wise linear embedding of $\boldsymbol{x}$: $\phi_{node}(\boldsymbol{x})$. The PPE is the linear projection ϕ_{PPE} of relative positions with respect to the probing port node $\boldsymbol{x}_{\text{probe}}$: $\phi_{\text{PPE}}\left(\{||\boldsymbol{x}_i - \boldsymbol{x}_{\text{probe}}||\}_{i=1}^N\right)$ as shown in Fig. 3b. The sum of processed node embeddings and PPE is then fed into the L-layered MHA and FF to generate $\boldsymbol{h}$[3]:

$$\boldsymbol{h}^0 \leftarrow \phi_{\text{node}}(\boldsymbol{x}) + \phi_{\text{PPE}}(\{||\boldsymbol{x}_i - \boldsymbol{x}_{\text{probe}}||\}_{i=1}^N)$$
$$\boldsymbol{h}^{l+1} \leftarrow \text{BN}\left(\Phi_{\text{FF}}\left(\text{BN}\left(\Phi_{\text{MHA}}\left(\boldsymbol{h}^l\right)\right)\right)\right)$$

where BN stands for batch normalization.

Decoder The decoder $p_{\text{dec}}(\boldsymbol{a}_t|\boldsymbol{h}, \boldsymbol{a}_{t-1})$ is composed of a single layer MHA to generate the action probability distribution with SoftMax. We improve generalization capabilities by capturing contextual information: we generate the query key Q via a context query embedding divided into a probing port context network (PCN ●) and recurrent context network (RCN ●), as shown in Fig. 3c. The PCN focuses on processing the embedding of probing port $\boldsymbol{h}_{\text{probe}}$ where RCN recurrently focuses on the embedding of previously selected action $\boldsymbol{h}_{\boldsymbol{a}_{t-1}}$:

$$\phi_{\text{PCN}}(\boldsymbol{h}_{\text{probe}}) = \text{MLP}(\boldsymbol{h}_{\text{probe}})$$
$$\phi_{\text{RCN}}(\boldsymbol{h}_{\boldsymbol{a}_{t-1}}) = \text{MLP}(\boldsymbol{h}_{\boldsymbol{a}_{t-1}})$$
$$Q = \phi_{\text{Linear}}(\phi_{\text{PCN}}(\boldsymbol{h}_{\text{probe}}) + \phi_{\text{RCN}}(\boldsymbol{h}_{\boldsymbol{a}_{t-1}}))$$

Then, the probability distribution is computed with MHA and linear projection ϕ_{Linear}:

$$p_{\text{dec}}(\boldsymbol{a}_t|\boldsymbol{h}, \boldsymbol{a}_{t-1}) = \text{SoftMax}\left(\phi_{\text{Linear}}\left((\text{MHA}\left(Q, K, V\right)\right)\right))$$

where K and V are a linear projection of $\boldsymbol{h}$.

3.2. Action-permutation Symmetricity and Order Bias

The symmetricity found in placement problems is the action-permutation (AP)-symmetricity, i.e., placement order does not affect the design performance. Let us denote t_i as a permutation of an action sequence $\{1, ..., K\}$, where K is the length of the action sequence. We then define the AP-transformation $T_{\text{AP}} = \{t_i\}_{i=1}^{K!}$ as a set of all possible permutations. The AP-symmetricity of DPP is the property that we want to impose on the design solver.

Definition 1 (AP-symmetricity). *For any $\boldsymbol{a} \in \mathcal{A}$, $\boldsymbol{x} \in \mathcal{X}$, $t \in T_{AP}$, the following holds:*

- *A scalar-valued function $f : \mathcal{A} \times \mathcal{X} \rightarrow \mathbb{R}$ is AP-symmetric if $f(\boldsymbol{a}, \boldsymbol{x}) = f(t(\boldsymbol{a}), \boldsymbol{x})$.*
- *A conditional probability distribution π is AP-symmetric if $\pi(\boldsymbol{a}|\boldsymbol{x}) = \pi(t(\boldsymbol{a})|\boldsymbol{x})$.*

$\mathcal{A}$ is the solution space and $\mathcal{X}$ is the task-condition space.

The objective function $\mathcal{J} : \mathcal{A} \times \mathcal{X} \rightarrow \mathbb{R}$ of DPP is an AP-symmetric function because $\boldsymbol{a}$ and $t(\boldsymbol{a})$ have identical placement design. AP-symmetricity can thus be induced to the policy π (conditional probability) to reflect the AP-symmetricity of an objective function $\mathcal{J}$. Moreover, we define an order bias metric, $b(\pi; \boldsymbol{p})$, to measure the AP-symmetricity.

Definition 2 (Order bias on distributions $\boldsymbol{p}$). *For a conditional probability $\pi(\boldsymbol{a}|\boldsymbol{x})$, where $\boldsymbol{x} \in \mathcal{X}$ and $\boldsymbol{a} \in \mathcal{A}$, the order bias $b(\pi; \boldsymbol{p})$, where $\boldsymbol{p} = \{p_{\mathcal{X}}, p_{\mathcal{A}}, p_{T_{AP}}\}$, refers to:*

$$b(\pi; \boldsymbol{p}) = \mathbb{E}_{p_{\mathcal{X}}(x)}\mathbb{E}_{p_{\mathcal{A}}(\boldsymbol{a})}\mathbb{E}_{p_{T_{AP}}(t)}[||\pi(\boldsymbol{a}|\boldsymbol{x}) - \pi(t(\boldsymbol{a})|\boldsymbol{x})||_1]$$

Intuitively, the order bias $b(\pi; \boldsymbol{p})$ is a general property of a sequential solution generation scheme. It measures how much the solver $\pi(\boldsymbol{a}|\boldsymbol{x})$ has different probabilities of generating AP-symmetric solutions. The order bias metric holds for the following theorem:

Theorem 1. *A task-conditioned policy $\pi(\boldsymbol{a}|\boldsymbol{x})$ is AP-symmetric if and only if its order bias is zero ($b(\pi; \boldsymbol{p}) = 0$) while the distributions are non-zero, $p_{\mathcal{X}}(\boldsymbol{x}) > 0, p_{\mathcal{A}}(\boldsymbol{a}) > 0, p_{T_{AP}}(t) > 0$, for any $x \in \mathcal{X}$, $\boldsymbol{a} \in \mathcal{A}$ and $t \in T_{AP}$.*

We report the detailed proof of Theorem 1 in Appendix F.

3.3. Collaborative Symmetricity Exploitation (CSE)

To induce the AP-symmetricity to the trained DEVFORMER, and thus to improve its generalization capability and data efficiency in training with low-data regimes, we design a novel training framework called *collaborative*

[3]Note that we indicate $\boldsymbol{h}^L$ as a high dimensional embedding $\boldsymbol{h}$ for a simple notation.

symmetricity exploitation (CSE). Fig. 4 shows the overall scheme: CSE exploits symmetricity in both expert and self-generated data.

Expert Exploitation The major role of expert exploitation is to train high-quality symmetric contextualized policies for various task conditions x by leveraging offline expert data a^* from the offline expert dataset $D_{\text{exp}} = \{(x^{(i)}, a^{(i)*})\}_{i=1}^{N}$ with T_{AP}. T_{AP} transforms each offline expert data a^* for P times to *augment* the offline expert dataset to reflect the AP-symmetric nature of the placement task. Specifically, we randomly choose $\{t_1, ..., t_P\} \subset T_{\text{AP}}$ to generate $D_{\text{aug}} = \{(x^{(i)}, a^{(i)*}), (x^{(i)}, t_1(a^{(i)*})), ..., (x^{(i)}, t_P(a^{(i)*}))\}_{i=1}^{N}$. Then, $\mathcal{L}_{\text{Expert}}$ is expressed as a *teacher-forcing* imitation learning scheme with the augmented expert dataset D_{aug}.

$$\mathcal{L}_{\text{Expert}} = -\mathbb{E}_{\boldsymbol{a}^*, \boldsymbol{x} \sim D_{\text{aug}}}[log \pi_\theta(\boldsymbol{a}^* | \boldsymbol{x})] \tag{3}$$

Note that expert exploitation is expected to reduce order bias defined with the three uniform distributions; $\boldsymbol{x}$ of $\mathcal{U}_{D_{\text{exp}}}(\boldsymbol{x})$, $\boldsymbol{a}$ of $\mathcal{U}_{D_{\text{exp}}}(\boldsymbol{a})$, and t of $\mathcal{U}_{T_{\text{AP}}(t)}$.

Self-Exploitation While D_{aug} only contains expert quality data, self-exploitation uses self-generated data, whose quality is poor initially but improves over the training phase. Thus, the self-exploitation scheme is designed to induce the AP-symmetricity in a wider action space to achieve greater generalization capability.

$$\mathcal{L}_{\text{Self}} = \mathbb{E}_{\mathcal{U}_{\mathcal{X}}(\boldsymbol{x})} \mathbb{E}_{\pi_{\tilde{\theta}}(\cdot|\boldsymbol{x})} \mathbb{E}_{\mathcal{U}_{T_{\text{AP}}}(\boldsymbol{t})} [||\pi_{\tilde{\theta}}(\boldsymbol{a'}|\boldsymbol{x}) - \pi_\theta(t(\boldsymbol{a'})|\boldsymbol{x})||_1] \tag{4}$$

Formally, the self-exploitation loss is a special form of order bias defined based on the distributions, $\boldsymbol{x} \sim \mathcal{U}_{\mathcal{X}}$, $\boldsymbol{a} \sim \pi_{\tilde{\theta}}$ (current policy) and $t \sim \mathcal{U}_{T_{\text{AP}}}$, where $\mathcal{U}$ is a uniform distribution; $\mathcal{L}_{\text{Self}} = b(\pi_\theta, \boldsymbol{p} = \{\mathcal{U}_{\mathcal{X}}, \pi_{\tilde{\theta}}, \mathcal{U}_{T_{\text{AP}}}\})$.

Loss function We design the DEVFORMER loss term $\mathcal{L}$ consisting of expert exploitation loss $\mathcal{L}_{\text{Expert}}$ and self-exploitation loss $\mathcal{L}_{\text{Self}}$ to reduce order bias (Definition 2):

$$\mathcal{L} := \mathcal{L}_{\text{Expert}} + \lambda \mathcal{L}_{\text{Self}} \tag{5}$$

where λ is a hyperparameter that adjusts the weighted ratio between $\mathcal{L}_{\text{Expert}}$ and $\mathcal{L}_{\text{Self}}$.

4. Experimental Results

4.1. Dataset and Benchmark

Benchmark Description The DPP is an important design optimization task to improve the power integrity performance of hardware to find optimal design $\boldsymbol{a}$ of context $\boldsymbol{x}$ that maximizes objective value $\mathcal{J}(\boldsymbol{a}; \boldsymbol{x})$ with a limited number of decaps K. In this benchmark (1) we measure objective score with $K = 20$ and (2) we measure minimum K for satisfying target objective $J = J^*$. Also, we evaluate the sample efficiency both for training time and test time. For the training time efficiency, we measure the number of offline datasets N. For the test time efficiency, the number of simulation shots M.

Baselines We report various existing baselines for DPP devised by hardware domain researchers. We categorize them into four categories:

- **Test Time Search.** A traditional search method of random search and genetic algorithm (GA) on the test time are benchmarked.
- **Online Test Time Adaptation.** Online DRL methods for DPP that directly solve test problems with the learning procedure are reported: CNN deep Q learning (Park et al., 2020b, CNN-DQN), CNN double DQN (Zhang et al., 2020, CNN-DDQN), pointer network policy gradient (Kim et al., 2021, Pointer-PG), and AM policy gradient (Park et al., 2020a, AM-PG).
- **Online Contextual Pretraining.** Existing online contextual DRL methods for DPP, which amortize the search process by pretraining are reported: pointer network with contextual RL (Kim et al., 2021, Pointer-CRL) and (Park et al., 2022b, AM-CRL).
- **Offline Contextual Pretraining.** Imitation learning-based contextual methods for DPP which amortize the search process with the expert dataset are reported: Pointer-CIL and AM-CIL.

Dataset and Simulator For the DPP score simulator, a chip-package hierarchical PDN is used. The PDN model is represented as a $N_{row} \times N_{col} = 10 \times 10$ grid over 201 frequency points linearly distributed between 200MHz and 20GHz, which gives $100 \times 100 \times 201 \approx 2M$ impedances to be evaluated per each task; simulation intensive.

We use N expert data, collected by a genetic algorithm (GA) with a specific number of simulations $M = 100$ per each. For the test dataset of performance evaluation, 100 PDN cases are used. See Appendix A.2 and Appendix A.5 for detailed data construction.

Implementation For DevFormer, we use encoder layers of $L = 3$ and 128 hidden dimensions of MHA, and 512 hidden dimensions for feed-forward. See Appendix C.1 for a detailed setup of training hyperparameters. For the implementation of baseline, methods see Appendix C.2 and Appendix C.3 for details.

Table 1: Performance evaluation results. We report the number of shots and average score of 100 test problems. Each method was sweeped for 5 different seeds and the average score and S.D. are reported.

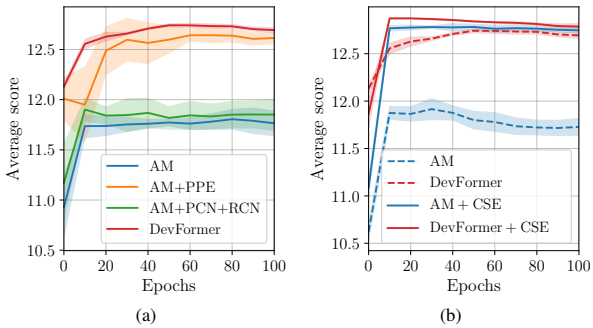
Method	Num. Shot (M) ↓	Score ↑
Online Test Time Search		
GA (*expert*)	100	12.56 ± 0.017
RS	10,000	12.70 ± 0.000
Online Test Time Adaptation		
Pointer-PG	10,000	9.66 ± 0.206
AM-PG	10,000	9.63 ± 0.587
CNN-DQN	10,000	9.79 ± 0.267
CNN-DDQN	10,000	9.63 ± 0.150
Online Contextual Pretraining & Zero Shot Inference		
Pointer-CRL	Zero Shot	9.59 ± 0.232
AM-CRL	Zero Shot	9.56 ± 0.471
Offline Contextual Pretraining & Zero Shot Inference		
Pointer-CIL	Zero Shot	10.49 ± 0.119
AM-CIL	Zero Shot	11.74 ± 0.075
DevFormer-CSE	**Zero Shot**	$\mathbf{12.88 \pm 0.003}$

4.2. Performance Evaluation

For performance evaluation, we set $N = 2000$ for the offline pretraining and $M > 200000$ for online pretraining shots. The pretraining methods are evaluated with zero-shot inference at the test time.

As shown in Table 1, our DevFormer significantly outperformed all baselines in terms of average performance score. See Eq. (S1) in Appendix A.4 for the performance metric. Online search methods generally find solutions that give a high average performance. This is due to a large number of searching iterations M, which incurs the same number of costly simulations. On the other hand, the contextual pretrained methods including DevFormer do not require simulations to generate solutions; once trained, they only require a single simulation to measure the performance after zero shot inference. DevFormer is the only learning-based method capable of finding a solution that outperforms the highly iterative online search methods even by a zero-shot inference.

The online DRL methods showed poor performance in general. When the number of costly simulations was limited, RL-based methods (AM-CRL, Pointer-CRL) showed poorer generalization capability than their IL versions (AM-CIL, Pointer-CIL) due to inefficiency in exploring over extremely large combinatorial action space of DPP. We believe that the imitation learning approach, fitting the policy with offline expert data, has greater exploration capability with the help of expert policy thus able to achieve higher performance with a limited simulation budget (see Appendix C.2).



(a) (b)

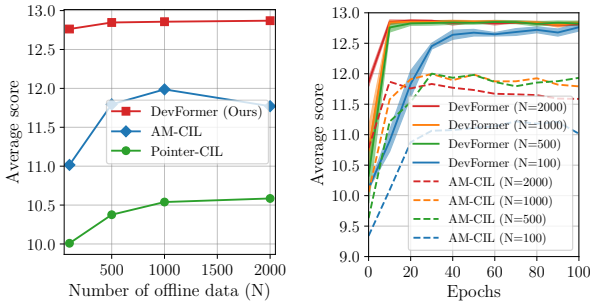
Figure 5: (a) Ablation on DevFormer components without CSE. Note that DevFormer architecture is based on AM-CIL with newly devised PPE, PCN and RCN. AM refers to AM-CIL. (b) Ablation on AM and DevFormer architecture with and without CSE learning scheme.

Among the CIL approaches, DevFormer showed the highest performance. We believe that such higher zero-shot performance comes from both symmetricity exploitation schemes and the newly devised neural architecture: (1) expert exploitation and self-exploitation with symmetric label transformation amplify the number of data to train with and induce solution symmetricity to improve generalization capability. (2) PPE, PCN, and RCN in DevFormer make the policy easily adapt to new task conditions.

Extrapolation over Expert Method The DevFormer is trained with the offline expert data generated by the GA outperformed the GA. That is, the DevFormer policy trained with lower-quality data produces higher-quality designs. We believe this is possible because we trained a factorized form of the policy that does not predict labels in a single step but produced a solution through a serial iterative roll-out process, during which a good strategy for placing decaps can be identified.

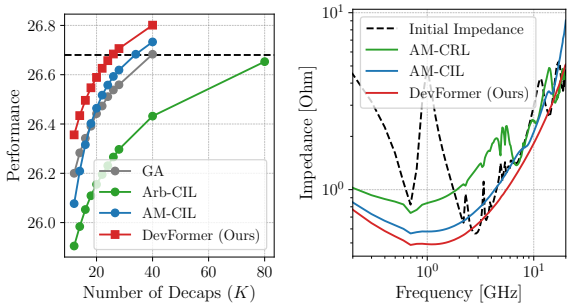
4.3. Ablation Study

Effectiveness of DevFormer Components and CSE. We conducted ablation studies to verify the effectiveness of the proposed DevFormer neural architecture and CSE training scheme. A detailed ablation on DevFormer components without CSE is reported in Fig. 5a. The addition of PPE, PCN and RCN on top of AM-CIL baseline showed clear performance improvement. Furthermore, in Fig. 5b, both AM and DevFormer neural architectures perform better with the presence of CSE and DevFormer always outperforms the AM despite the presence of CSE. Thus, we verified that both CSE and DevFormer contribute to the performance improvement.



(a) (b)

Figure 6: Sample efficiency of DEVFORMER in comparison to other CIL methods. (a) Performance evaluation depending on the number of offline data used for training. (b) Training convergence graph depending on the number of offline data used for training.



(a) (b)

Figure 7: (a) Performance comparison with the number of decap variation on HBM PDN. (b) Magnitude of resulting impedance suppression over wide frequency range after decap placement.

Table 2: Valuation of Order Bias

	$b(\pi, \boldsymbol{p} = \{\mathcal{U}_{\mathcal{X}}, \pi, \mathcal{U}_{T_{AP}}\})$
AM-CIL	8.70×10^{-21}
DEVFORMER	$\mathbf{1.25 \times 10^{-28}}$

Order Bias Measurement To empirically prove that our CSE induces AP-symmetricity, we measured $b(\pi, \boldsymbol{p} = \{\mathcal{U}_{\mathcal{X}}, \pi, \mathcal{U}_{T_{AP}}\}$ for sample width 100 and took the average value. As shown in Table 2, our CSE significantly reduced the order bias defined in Definition 2, verifying that CSE successfully induced the AP-symmetricity.

4.4. Sample Efficiency Evaluation for Offline Dataset

We investigated how the number N of offline data generated by the GA affects the performance of DEVFORMER, compared to AM-CIL and Pointer-CIL. As shown in Fig. 6, DEVFORMER outperforms the baselines in all N variation; DEVFORMER trained with $N = 100$ even outperformed the baselines trained with $N = 2000$.

Table 3: Scalability evaluations on larger power distribution network (PDN) scale and a varying number of deep K. The scale $\times$ scale indicates the size of input grids for PDN. K refers to the number of decap placed on the target PDN. Scores in bold indicate the best scores. A lower K with a higher score value indicates a Pareto score.

Scale Variables		Methods		
PDN	K	GA	AM-CIL	Ours
10×10	12	11.77	10.22	**12.23**
	16	12.25	11.13	**12.60**
	20	12.53	11.71	**12.81**
	24	12.79	12.20	**12.95**
	30	13.02	12.62	**13.11**
15×15	20	7.61	6.23	**8.47**
	40	7.69	7.75	**8.54**

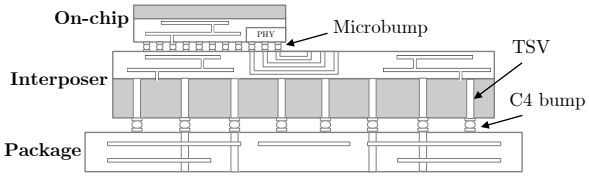


Figure 8: Structure of a three-layer HBM PDN model.

4.5. Zero shot Generalization to various tasks

To verify zero shot capability on various scales of tasks, learning-based DPP methods were pretrained for a fixed scale PDN (10×10) and a fixed number of decaps, $K = 20$. Then, the pretrained models are asked to place decaps of varying K on (10×10) PDN and a larger (15×15) PDN without additional training (i.e., zero-shot). As shown in Table 3, our DEVFORMER outperformed GA and AM-CIL for all scales. Furthermore, DEVFORMER achieved greater performance with fewer decaps. Reducing the number of decaps has a significant industrial impact; as hardware devices are mass-produced, reducing a single decap saves enormous fabrication costs.

4.6. Application on Real-world Hardware

To verify the practical applicability of the DEVFORMER, we applied the proposed DEVFORMER to a real-world hardware application, the high bandwidth memory (HBM), which is an interposer-based 2.5D IC. As shown in Fig. 8, the hierarchical PDN model of HBM is composed of (40×40) package PDN, (40×60) interposer PDN and (15×20) on-chip PDN, each layer connected by TSV + C4 bumps and microbumps.

For performance evaluation, we compared DEVFORMER to GA, AM-CIL, and Pointer-CIL on placing a varying number of decaps, K, for 100 unseen test cases. The pretrained solvers with $K = 20$ were used for DEVFORMER, AM-CIL, and Pointer-CIL without additional training.

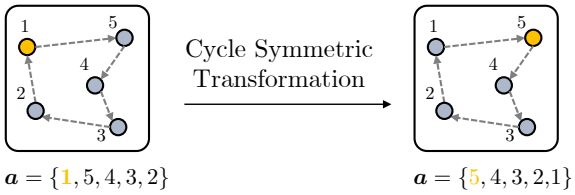
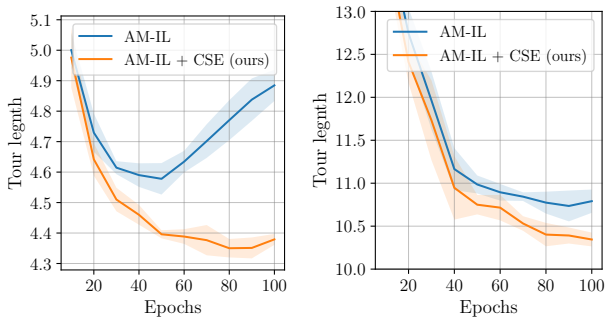


Figure 9: Example of cycle symmetricity in the traveling salesman problem (TSP).



(a) TSP ($N = 20$) (b) TSP ($N = 100$)

Figure 10: CSE application on TSP cyclic symmetricity. AM-IL stands for the imitation learning trained AM with sparse expert labels. The CSE greatly improves the performance of AM-IL as TSP ($N = 20$) and ($N = 100$).

The results in Fig. 7a demonstrate that DEVFORMER achieves higher performance with significantly fewer decaps than other methods. The performance score of 26.68, which was attained with 40 decaps by the GA and 34 decaps by AM-CIL method, was achieved with only 26 decaps by DEVFORMER in a zero-shot setting. The pointer-CIL method could not achieve the same performance score even with 80 decaps.

The resulting impedance over a wide frequency range after decap placement by each method is illustrated in Fig. 7b. It shows that decap placement by DEVFORMER suppressed the impedance the most, leading to greater power integrity and objective value.

We also conducted a power noise analysis on a test case (see Appendix D.3). DEVFORMER was able to reduce the power noise by 94.2% using only 26 decaps, while GA reduced by 93.5% using 40 decaps. As hardware devices are mass-produced, reducing even a single decap can greatly reduce production costs (Koo et al., 2018). With a more than 30% reduction in the number of decaps compared to the best-performing baseline, our DEVFORMER can significantly contribute to the industry.

4.7. Application to Other Offline Contextual Designs

Our CSE learning scheme induces AP-symmetricity in the DEVFORMER neural architecture and can also be applied to other architectures for offline contextual design. We tested its versatility by implementing it on the AM transformer model (Kool et al., 2019) to solve the traveling salesman problem (TSP), a combinatorial optimization problem whose objective is to find the optimal tour sequence that visits all cities with minimum tour length. CSE is used to enforce cyclic symmetricity in the TSP solutions by considering the N (the number of cities) symmetric solutions that can be obtained by permuting the initial cities, where the tour length is invariant (see Fig. 9).

We conducted a benchmark using a TSP problem with sparse expert data, using only 100 labels derived from the TSP Concorde (Applegate et al., 2006) solver. As shown in Fig. 10, CSE significantly improves the performance of AM on 100 randomly generated synthetic test data. This indicates that our newly devised CSE can be further applied to various domains that require solution symmetricity, e.g., molecular generation, considering the symmetric nature of the molecular graph (Bengio et al., 2021).

5. Related Works

Symmetricity Learning in Solution Space. Several studies have leveraged symmetricity in solution space. POMO (Kwon et al., 2020) proposed a reinforcement learning scheme that leverages symmetricity in TSP, using the cyclic property that identical solutions can be expressed as multiple permutations of initially visited nodes. GFlowNet (Bengio et al., 2021) uses a generative flow to train policy distributions proportional to reward distributions and applies a directed acyclic graph (DAG) instead of a classical tree structure to induce symmetricity. Our method is similar to POMO in that it leverages symmetricity in the solution space through regularization, but it applies this approach to imitation learning.

New Transformers. There have been numerous efforts to adapt the transformer architecture (Vaswani et al., 2017) to new domains beyond natural language processing (NLP) (Brown et al., 2020). Examples include the vision transformer (ViT) (Dosovitskiy et al., 2020), graph transformer (Graphormer) (Ying et al., 2021), attention model for combinatorial optimization (AM) (Kool et al., 2019), and decision transformer for reinforcement learning (DT) (Chen et al., 2021). These efforts aim to incorporate domain-specific knowledge and remove unnecessary priors from the original transformer architecture, which was primarily designed for sequential data and language modeling. Our proposed method is a novel transformer for device optimization in hardware and is part of this ongoing research trend.

6. Conclusion

In this paper, we presented DEVFORMER, a novel transformer model for solving contextual offline hardware design problems, and CSE, a learning scheme that induces AP-symmetricity to the neural architecture. By incorporating strong domain-specific inductive biases, our model learns more efficiently and effectively, overcoming the limitations of traditional transformer architectures. We have demonstrated the proposed approach on a novel hardware benchmark and validated its extensibility to other combinatorial optimization problems. DEVFORMER achieved higher performance compared to other methods while considerably reducing production costs on a real-world high bandwidth memory. Future works include exploring its applicability to other hardware design tasks and investigating its scalability to larger and more complex design problems.

Acknowledgement

We thank Hyeonah Kim, Hyunwook Park, Subin Kim, and the anonymous reviewers for their invaluable contributions in providing insightful feedback on our manuscript.

References

Agnesina, A., Chang, K., and Lim, S. K. Vlsi placement parameter optimization using deep reinforcement learning. In *Proceedings of the 39th International Conference on Computer-Aided Design*, ICCAD '20, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450380263. doi: 10.1145/3400302.3415690. URL https://doi.org/10.1145/3400302.3415690.

Applegate, D., Bixby, R., Chvatal, V., and Cook, W. Concorde tsp solver. 2006. URL http://www.math.uwaterloo.ca/tsp/concorde/m.

Bello, I. et al. Neural combinatorial optimization with reinforcement learning. *CoRR*, abs/1611.09940, 2016. URL http://arxiv.org/abs/1611.09940.

Bengio, E., Jain, M., Korablyov, M., Precup, D., and Bengio, Y. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394, 2021.

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A., and Mordatch, I. Decision transformer: Reinforcement learning via sequence modeling. *Advances in neural information processing systems*, 34:15084–15097, 2021.

Cheng, R. and Yan, J. On joint learning for solving placement and routing in chip design. *Advances in Neural Information Processing Systems*, 34, 2021.

de Paulis, F., Cecchetti, R., Olivieri, C., and Buecker, M. Genetic algorithm pdn optimization based on minimum number of decoupling capacitors applied to arbitrary target impedance. In *2020 IEEE International Symposium on Electromagnetic Compatibility Signal/Power Integrity (EMCSI)*, pp. 428–433, 2020. doi: 10.1109/EMCSI38923.2020.9191458.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2020.

Erdin, I. and Achar, R. Multi-objective optimization of decoupling capacitors for placement and component value. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 9(10):1976–1983, 2019. doi: 10.1109/TCPMT.2019.2930565.

Fan, J., Knighten, J., Orlandi, A., Smith, N., and Drewniak, J. Quantifying decoupling capacitor location. In *IEEE International Symposium on Electromagnetic Compatibility. Symposium Record (Cat. No.00CH37016)*, volume 2, pp. 761–766 vol.2, 2000. doi: 10.1109/ISEMC.2000.874717.

Farrahi, S. and Koether, E. Effect of power plane inductance on power delivery networks. In *DesignCon*, 2019.

Haaswijk, W., Collins, E., Seguin, B., Soeken, M., Kaplan, F., Süsstrunk, S., and De Micheli, G. Deep learning for logic optimization algorithms. In *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, 2018. doi: 10.1109/ISCAS.2018.8351885.

Hallak, A., Di Castro, D., and Mannor, S. Contextual markov decision processes. *arXiv preprint arXiv:1502.02259*, 2015.

Hosny, A., Hashemi, S., Shalan, M., and Reda, S. Drills: Deep reinforcement learning for logic synthesis. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 581–586, 2020. doi: 10.1109/ASP-DAC47756.2020.9045559.

Hwang, J., Pak, J. S., Yoon, D., Lee, H., Jeong, J., Heo, Y., and Kim, I. Enhancing on-die pdn for optimal use of package pdn with decoupling capacitor. In

2021 IEEE 71st Electronic Components and Technology Conference (ECTC), pp. 1825–1830, 2021. doi: 10.1109/ECTC32696.2021.00288.

Juang, J., Zhang, L., Kiguradze, Z., Pu, B., Jin, S., and Hwang, C. A modified genetic algorithm for the selection of decoupling capacitors in pdn design. In *2021 IEEE International Joint EMC/SI/PI and EMC Europe Symposium*, pp. 712–717, 2021. doi: 10.1109/EMC/SI/PI/EMCEurope52599.2021.9559292.

Kim, H., Park, H., Kim, M., Choi, S., Kim, J., Park, J., Kim, S., Kim, S., and Kim, J. Deep reinforcement learning framework for optimal decoupling capacitor placement on general pdn with an arbitrary probing port. In *2021 IEEE 30th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, pp. 1–3, 2021. doi: 10.1109/EPEPS51341.2021.9609194.

Kim, J. et al. Chip-package hierarchical power distribution network modeling and analysis based on a segmentation method. *IEEE Transactions on Advanced Packaging*, 33(3):647–659, 2010. doi: 10.1109/TADVP.2010.2043673.

Koo, K., Luevano, G. R., Wang, T., Özbayat, S., Michalka, T., and Drewniak, J. L. Fast algorithm for minimizing the number of decap in power distribution networks. *IEEE Transactions on Electromagnetic Compatibility*, 60(3):725–732, 2018. doi: 10.1109/TEMC.2017.2746677.

Kool, W., van Hoof, H., and Welling, M. Attention, learn to solve routing problems! In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=ByxBFsRqYm.

Kwon, Y.-D., Choo, J., Kim, B., Yoon, I., Gwon, Y., and Min, S. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

Liao, H., Zhang, W., Dong, X., Poczos, B., Shimada, K., and Burak Kara, L. A Deep Reinforcement Learning Approach for Global Routing. *Journal of Mechanical Design*, 142(6), 11 2019. ISSN 1050-0472. doi: 10.1115/1.4045044. URL https://doi.org/10.1115/1.4045044. 061701.

Liao, H., Dong, Q., Qi, W., Fallon, E., and Kara, L. B. Track-assignment detailed routing using attention-based policy model with supervision. In *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, pp. 105–110, 2020.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Nazi, A., Pak, J., Tong, A., Srinivasa, K., Hang, W., Tuncer, E., Le, Q., Laudon, J., Ho, R., Carpenter, R., and Dean, J. A graph placement methodology for fast chip design. *Nature*, 594:207–212, 06 2021. doi: 10.1038/s41586-021-03544-w.

Park, H., Kim, M., Kim, S., Jeong, S., Kim, S., Kang, H., Kim, K., Son, K., Park, G., Son, K., Shin, T., and Kim, J. Policy gradient reinforcement learning-based optimal decoupling capacitor design method for 2.5-d/3-d ics using transformer network. In *2020 IEEE Electrical Design of Advanced Packaging and Systems (EDAPS)*, pp. 1–3, 2020a. doi: 10.1109/EDAPS50281.2020.9312908.

Park, H., Park, J., Kim, S., Cho, K., Lho, D., Jeong, S., Park, S., Park, G., Sim, B., Kim, S., Kim, Y., and Kim, J. Deep reinforcement learning-based optimal decoupling capacitor design method for silicon interposer-based 2.5-d/3-d ics. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 10(3):467–478, 2020b. doi: 10.1109/TCPMT.2020.2972019.

Park, H., Kim, M., Kim, S., Kim, K., Kim, H., Shin, T., Son, K., Sim, B., Kim, S., Jeong, S., Hwang, C., and Kim, J. Transformer network-based reinforcement learning method for power distribution network (pdn) optimization of high bandwidth memory (hbm). *IEEE Transactions on Microwave Theory and Techniques*, 70 (11):4772–4786, 2022a. doi: 10.1109/TMTT.2022.3202221.

Park, H., Kim, M., Kim, S., Kim, K., Kim, H., Shin, T., Son, K., Sim, B., Kim, S., Jeong, S., Hwang, C., and Kim, J. Transformer network-based reinforcement learning method for power distribution network (pdn) optimization of high bandwidth memory (hbm). *IEEE Transactions on Microwave Theory and Techniques*, 70 (11):4772–4786, 2022b. doi: 10.1109/TMTT.2022.3202221.

Swaminathan, M. and Engin, A. *Power Integrity Modeling and Design for Semiconductors and Systems*. Prentice Hall Press, USA, first edition, 2007. ISBN 9780136152064.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 30, pp. 5998–6008. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Vinyals, O., Fortunato, M., and Jaitly, N. Pointer networks. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 28. Curran Associates, Inc., 2015. URL https://proceedings.neurips.cc/paper/2015/file/29921001f2f04bd3baee84a12e98098f-Paper.pdf.

Xu, Z., Wang, Z., Sun, Y., Hwang, C., Delingette, H., and Fan, J. Jitter-aware economic pdn optimization with a genetic algorithm. *IEEE Transactions on Microwave Theory and Techniques*, 69(8):3715–3725, 2021. doi: 10.1109/TMTT.2021.3087188.

Ying, C., Cai, T., Luo, S., Zheng, S., Ke, G., He, D., Shen, Y., and Liu, T.-Y. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems*, 34:28877–28888, 2021.

Zhang, L., Huang, W., Juang, J., Lin, H., Tseng, B.-C., and Hwang, C. An enhanced deep reinforcement learning algorithm for decoupling capacitor selection in power distribution network design. In *2020 IEEE International Symposium on Electromagnetic Compatibility Signal/Power Integrity (EMCSI)*, pp. 245–250, 2020. doi: 10.1109/EMCSI38923.2020.9191512.

Zhao, Z. and Zhang, L. Deep reinforcement learning for analog circuit sizing. In *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2020. doi: 10.1109/ISCAS45731.2020.9181149.

Appendix for "DevFormer: A Symmetric Transformer for Context-Aware Device Placement"

A. DPP Electrical Modeling and Problem Definition

This section provides electrical modeling details of PDN and decap models used for verification of DEVFORMER in DPP. Note that these electrical models can be substituted by those of the reader's interest. There are three methods to extract PDN and decap models that are also used for objective evaluation; 3D EM simulation tool, ADS circuit simulation tool, and unit-cell segmentation method. For each method, there exists a trade-off between time complexity and accuracy. See Table 1. Out of the three methods, we used the unit-cell segmentation method for a benchmark. Simulation time was evaluated using the same PDN model on a machine equipped with a 40 threads Intel© Xeon Gold 6226R CPU and 512GB of RAM. Note that simulation time depends on the size and the structural complexity of the PDN model.

Table 1: Time Taken for an Objective Evaluation of a PDN model described in Appendix A.2

Simulation Method	Time Taken
EM Simulation Tool	≈10 hours
ADS Circuit Simulation Tool	23.58 sec

A.1. Domain Perspective Decap Placement Problem

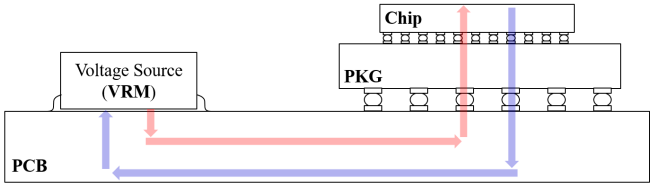
The development of AI has led to an increased demand for high-performance computing systems. High-performance computing systems not only require the precise design of hardware chips such as CPU, GPU, and DRAM but also require stable delivery of power to the operating integrated circuits. Power delivery has become a huge technical bottleneck of hardware devices due to the continuously decreasing supply voltage margin along with the technology shrink of the transistors (Hwang et al., 2021).

Fig. S1 (a) shows the power distribution network (PDN) consisting of all the power/ground planes from the voltage source to operating chips. Power is generated in the voltage regulator module (VRM) and delivered through electrical interconnections of PCB, package and chip. Finding ways to meet the desired voltage and current from the power source to destinations along the PDN is detrimental because failure in achieving power integrity (PI) leads to various reliability problems such as incorrect switching of transistors, crosstalk from neighboring signals, and timing margin errors (Swaminathan & Engin, 2007). Decoupling capacitors (decaps) placed on the PDN allows a reliable power supply to the operating chips, thus improving the power integrity of hardware. As shown in Fig. S1 (b)-(c), the role of decap is analogous to that of water storage tanks, placed along the city, apartment, and household, that can provide water uninterruptedly and reliably. As placing more water tanks can make the water supply more stable, placing more decaps can make the power supply more reliable. However, because adding more decaps requires more space and is costly, optimal placement of decaps is important in terms of PI and cost/space-saving.

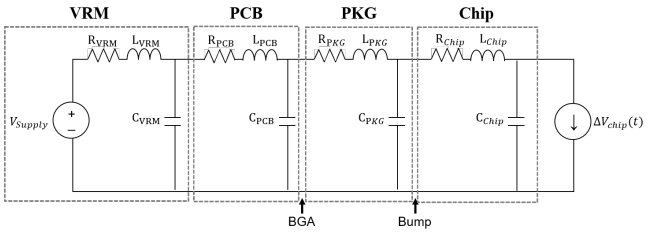
A.2. PDN and Decap Models for Verification

Unit-Cell Segmentation Method. The segmentation method (Kim et al., 2010) is a simple and fast way to generate approximated electrical models. Because the analysis of the full electrical model using EM simulation is very time-consuming, we divided the full PDN model into smaller unit-cells and constructed the full PDN model using the unit-cell segmentation method. For fast simulation, we used an equation-based implementation in Python of the segmentation method, illustrated in Fig. S2.

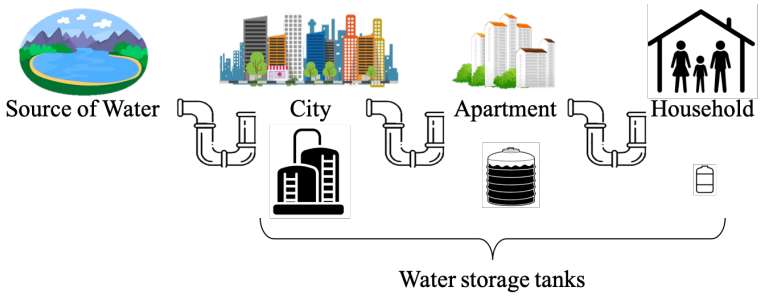
The segmentation method was used for the generation of the PDN model consisting of a chip layer and a package layer for verification as illustrated in Fig. S2 (a). The segmentation method was also used for the objective evaluation of DPP. When a solution for DPP is made, decaps are placed on the corresponding ports on PDN using the segmentation method as illustrated in Fig. S2 (b).



(a) An example of hierarchical power distribution network (PDN).



(b) Electrical circuit model of the hierarchical PDN in (a).



(c) Water supply chain from the source to household.

Figure S1: Illustration of Hierarchical Power Distribution Network (PDN) analogous to Water Supply Chain. Similarly to how placing more water tanks can make the water supply more stable, placing more decaps can make the power supply more reliable

.

The PDN model we used for verification has a two-layer structure; a package layer at the bottom and a chip layer on top of it as illustrated in Fig. S3. The PDN was modeled through the unit-cell segmentation method. The package layer was composed of 40×40 package unit-cells and the chip layer was composed of 10×10 (i.e, $N_{row} \times N_{col}$) chip unit-cells. Because the DPP benchmark places MOS-type decaps, which are placed on the chip, ports are only available on the chip. Thus, we extracted 10×10 ports information from the chip layer. See Fig. S6 (a), illustrating the chip PDN divided into 10×10 units and each unit-cell numbered.

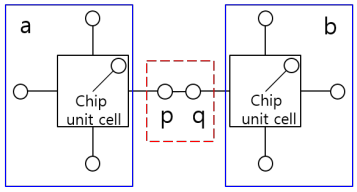
The electrical models of package and chip unit-cells that are used to build the PDN model for verification are described in Fig. S4. The chip layer is composed of 10×10 unit-cells, and the package layer is composed of 40×40 unit-cells using the segmentation method. The corresponding values of electrical parameters are listed in Table 2.

Table 2: Width and Electrical Parameters for Chip and Package Unit-Cells used for Verification.

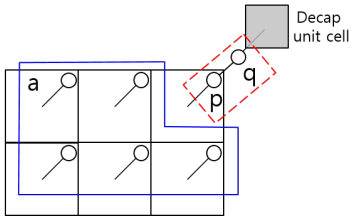
Unit-Cell Model	W	R	L	G	C
Chip	300μm	0.26 Ω	22pH	1.2mS	0.77pF
Package	0.5mm	0.093 Ω	0.25nH	5.4μS	0.045pF

We implemented MOS type decap for verification. The Decap model and its electrical parameters are shown in Fig. S5. As mentioned in Fig. S2 (b), the solution to DPP is evaluated using the segmentation method.

Note that these electrical parameters and PDN structures were used as a benchmark. For practical use of DEVFORMER, these PDN and decap models can be substituted by those of the reader's interest.



(a) Generation of Chip PDN.



(b) Decap Placement.

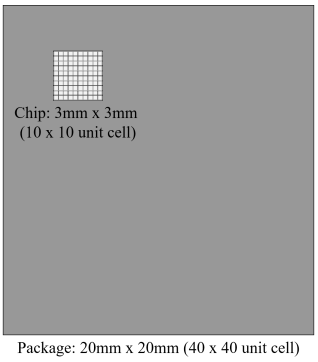
Boundary Conditions: $I_p = -I_q$, $V_p = V_q$

$$\begin{bmatrix} V_a \\ V_p \end{bmatrix} = \begin{bmatrix} Z_{aa} & Z_{ap} \\ Z_{pa} & Z_{pp} \end{bmatrix} \begin{bmatrix} I_a \\ I_p \end{bmatrix} \qquad \begin{bmatrix} V_b \\ V_q \end{bmatrix} = \begin{bmatrix} Z_{bb} & Z_{bq} \\ Z_{qb} & Z_{qq} \end{bmatrix} \begin{bmatrix} I_b \\ I_q \end{bmatrix}$$

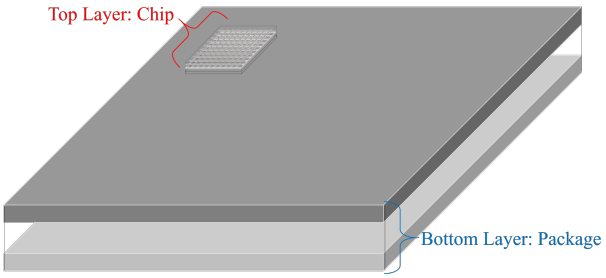
$$\begin{bmatrix} V_a \\ V_b \end{bmatrix} = \underbrace{\begin{bmatrix} Z_{aa} - Z_{ap}(Z_{pp} + Z_{qq})^{-1} Z_{pa} & Z_{ap}(Z_{pp} + Z_{qq})^{-1} Z_{qb} \\ Z_{bq}(Z_{pp} + Z_{qq})^{-1} Z_{pa} & Z_{bb} - Z_{bq}(Z_{pp} + Z_{qq})^{-1} Z_{qb} \end{bmatrix}}_{Z_{a \text{ and } b}} \begin{bmatrix} I_a \\ I_b \end{bmatrix}$$

(c) Segmentation Method.

Figure S2: Segmentation Method Implemented for PDN Generation and Decap Placement on PDN.



(a) Top-View of PDN model.

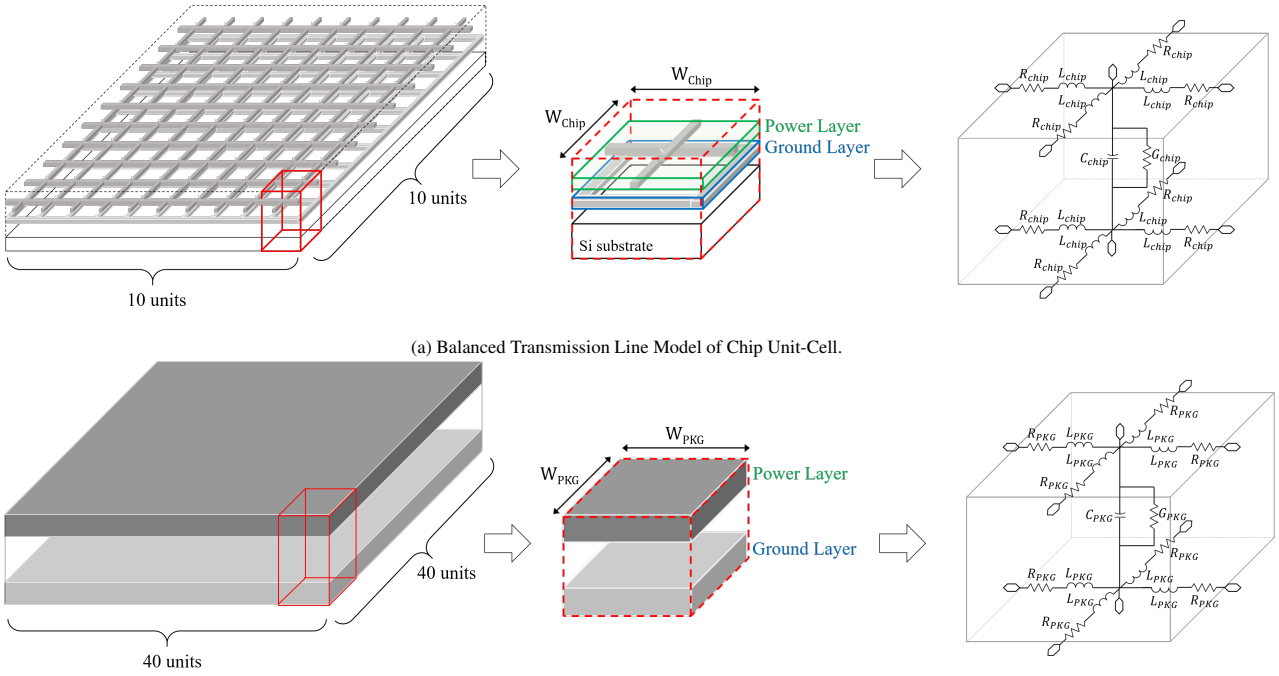


(b) Side-View of PDN model.

Figure S3: Top-view and Side-view of PDN Model used for Verification

A.3. Input Problem PDN and Output Decap Placement Data Structure

Each unit-cell (i.e, port) of the PDN model described in Appendix A.2 is represented as a set of 3D feature vectors composed of x-coordinate, y-coordinate and port condition; 1 representing the keep-out region, 2 representing a probing port and 0 for the decap allowed ports. Total 10×10 (i.e, $N_{row} \times N_{col}$) 3D vectors represent the problem PDN. The solution to DPP is the placement of decaps. As illustrated in Fig. S6 (b), the solution is given as a set of port numbers corresponding to each decap location.



(a) Balanced Transmission Line Model of Chip Unit-Cell.

(b) Balanced Transmission Line Model of Package Unit-Cell.

Figure S4: Electrical Modeling of Chip and Package Unit-Cells for PDN Model generation.

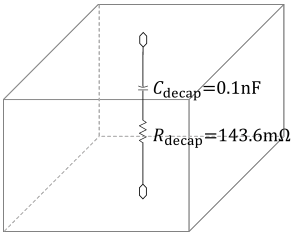


Figure S5: Decap Unit-Cell with the Electrical Parameters used for Verification.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}	x_{17}	x_{18}	x_{19}	x_{20}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}	x_{26}	x_{27}	x_{28}	x_{29}	x_{30}
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}	x_{36}	x_{37}	x_{38}	x_{39}	x_{40}
x_{41}	x_{42}	x_{43}	x_{44}	x_{45}	x_{46}	x_{47}	x_{48}	x_{49}	x_{50}
x_{51}	x_{52}	x_{53}	x_{54}	x_{55}	x_{56}	x_{57}	x_{58}	x_{59}	x_{60}
x_{61}	x_{62}	x_{63}	x_{64}	x_{65}	x_{66}	x_{67}	x_{68}	x_{69}	x_{70}
x_{71}	x_{72}	x_{73}	x_{74}	x_{75}	x_{76}	x_{77}	x_{78}	x_{79}	x_{80}
x_{81}	x_{82}	x_{83}	x_{84}	x_{85}	x_{86}	x_{87}	x_{88}	x_{89}	x_{90}
x_{91}	x_{92}	x_{93}	x_{94}	x_{95}	x_{96}	x_{97}	x_{98}	x_{99}	x_{100}

■ : Probing port ■ : Keep-Out Region

DPP: $x = \{(0,0,0), (\mathbf{0},\mathbf{1},\mathbf{1}), (0,2,0), \ldots, (3,7,2), (3,8,0), (\mathbf{3},\mathbf{9},\mathbf{1}), \ldots, (9,9,0)\}$

(a) Input Problem PDN.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50
51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70
71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90
91	92	93	94	95	96	97	98	99	

■ : Decap

Solution: $a = \{8, 12, 22, 25, 27, 28, 29, 32, 37, 39, 46, 48, 50, 59, 61, 62, 63, 68, 92, 97\}$

(b) Output Decap Placement Solution.

Figure S6: Illustration of how the DPP problem with specific conditions is given as an input and decap placement solution is generated as an output.

A.4. Objective Function of DPP

The objective of DPP is evaluated by power integrity (PI) simulation that computes the level of impedance suppression over a specified frequency domain:

$$\mathcal{J} := \sum_{f \in F} (Z_{initial}(f) - Z_{final}(f)) \cdot \frac{1\text{GHz}}{f} \tag{S1}$$

where $Z_{initial}$ and Z_{final} are the initial and final impedance at the frequency f before and after placing decaps, respectively. F is the set of specified frequency points. The more impedance is suppressed, the better the power integrity and the higher the performance score. Remark that DPP cannot be formulated as a conventional mixed-integer linear programming (MILP)-based combinatorial optimization because PI performance can not be formulated as a closed analytical form but can only be measured or simulated.

A.5. Random Problem Generation of DPP

To randomly generate decap placement problems (DPPs) with distinct conditions for training, testing and validation, a probing index I_{probe} is selected randomly from a uniform distribution of $\{1, ..., N_{row} \times N_{col}\}$. Then keep-out region indices $I_{keepout}$ are randomly selected through the following two stages: the number of keep-out regions $|I_{keepout}|$ is randomly selected from a uniform distribution of $0 \sim 15$. Then, a set of indices of keep-out ports $I_{keepout}$ is generated by random selection from the uniform distribution of $\{1, ..., N_{row} \times N_{col}\}$. We generated 100 test problems and 100 validation problems for 10×10 PDN and 50 test problems and 50 validation problems for 15×15 PDN. We made sure the training, test, and validation problems do not overlap.

B. Expert Label Collection

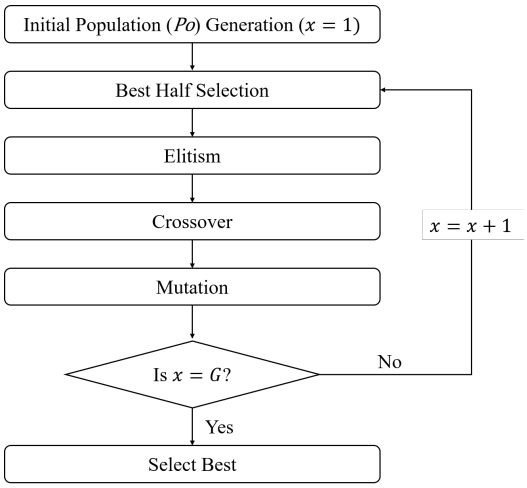
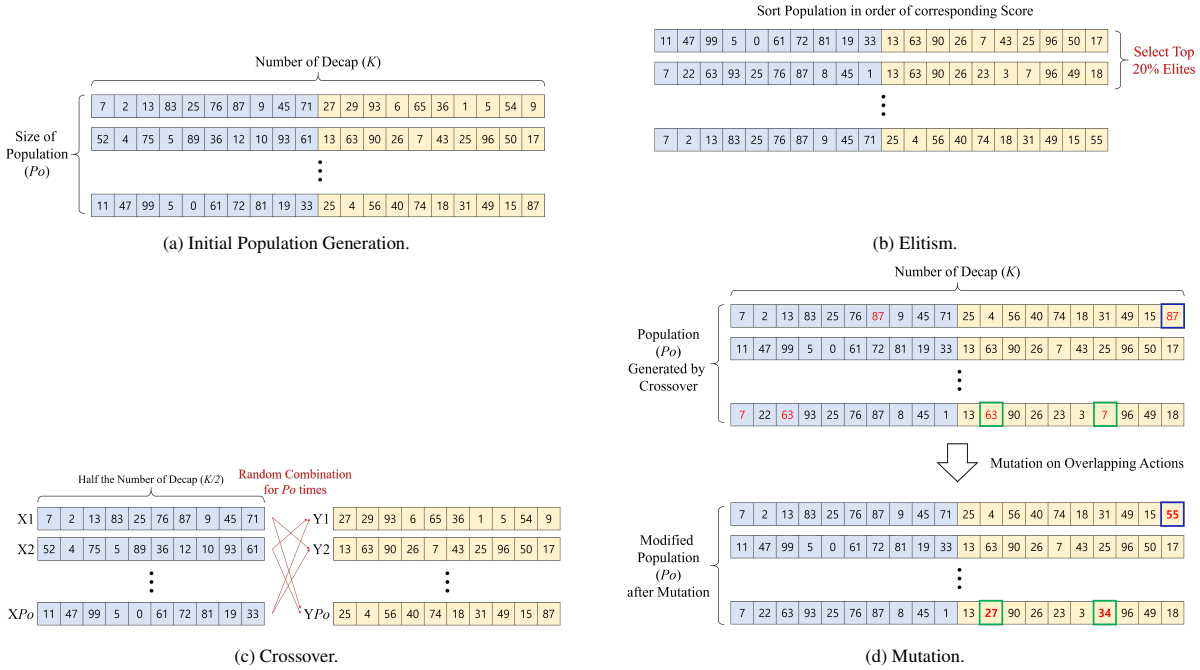


Figure S7: Process Flow of Genetic Algorithm for DPP.

We used a genetic algorithm (GA) as the expert policy to collect expert guiding labels for imitation learning. GA is the most widely used search heuristic method for DPP (Erdin & Achar, 2019; de Paulis et al., 2020; Xu et al., 2021; Juang et al., 2021). We devised our own GA for DPP, which aims to find the placement of a given number (K) of decaps on PDN with a probing port and 0-15 keep-out regions that best suppresses the impedance of the probing port.

Notations. M is the number of samples to undergo an objective evaluation to give the best solution. The value of M is defined by the size of population P_0 times the number of generation G. K refers to the number of decaps to be placed. P_{elite} is the number of elite population.

Guiding Dataset. To generate expert labels, guiding problems were generated in the same way the test dataset was generated. We ensured the guiding data problems do not overlap with the test dataset problems. Also, we made sure each guiding problem does not overlap. Each guiding data problem goes through the process described in Fig. S7 to collect the corresponding expert label.



(a) Initial Population Generation.

(b) Elitism.

(c) Crossover.

(d) Mutation.

Figure S8: Illustration of each GA Operators used for DPP Guiding Data Generation.

Population and Generation. For GA $\{M = 100\}$ (*expert policy*), we fixed the population size as $P_0 = 20$ and the generation number $G = 5$, which makes up the total number of samples to be $M = P_0 \times G = 100$. Each solution in the initial population is generated randomly. As described in Fig. S6 (b), each solution consists of K numbers, each representing a decap location on PDN. Note that each solution consists of random numbers from 0 to 99 except numbers corresponding to probing port and keep-out region locations.

Once the initial population is generated randomly, a new population is generated through elitism, crossover, and mutation. This whole process of generating a new population makes one generation; the generation process is iterated for $G - 1$ times.

Elitism. Once the initial population is formulated, the entire population undergoes objective evaluation and gets sorted in order of objective value. The size of elite population is pre-defined as $P_{elite} = 4$ for GA $\{M = 100\}$ (*expert policy*). That means the top 4 solutions in the population become the elite population and are kept for the next generation.

Crossover. Crossover is a process by which new population candidates are generated. Each solution of the current population, including the elites, is divided in half. Then, as described in Fig. S8 (c), half the solutions on the left and the other half on the right go through random crossover for P_0 times to generate a new population. If the elite population is available, $P_0 - P_{elite}$ random crossover takes place so that the total population size becomes P_0, including the elite population.

Mutation. According to Fig. S8 (d), solutions with overlapping numbers may exist after the random crossover. We replace the overlapping number with a randomly generated number, and we call this mutation.

Select Best. When G is reached, the final population is evaluated by the performance metric. Then, a solution with the highest objective value becomes the final guiding solution for the given DPP.

The guiding problems and corresponding solutions generated from GA are saved and used as guiding expert labels for imitation learning.

C. Detailed Experimental Settings

This section provides detailed experimental settings for main experiments and ablation studies.

C.1. Training Hyperparameters.

There are several hyperparameters for training; we tried to fix the hyperparameters as Kool et al. (2019) did to show their frameworks' practicality. We then provided several ablation studies on each hyperparameter to analyze how each component contributes to performance improvement.

Training hyperparameters are set to be identical to those presented in AM for TSP (Kool et al., 2019) except learning rate, unsupervised regularization rate λ, the number of expert data N, number of action permutation transformed data per expert data P and batch size B.

Table 3: Hyperparameter setting for training model.

Hyperparameter	Value
learning rate	10^{-5}
λ	$\times 10^{32}$
N	2000
P	4
B	100

C.2. Implementation of ML Baselines.

There are two main ML baselines, Pointer-CRL (Kim et al., 2021) and AM-CRL (Park et al., 2022a).

Pointer-CRL. Pointer-CRL is a PointerNet-based DPP solver proposed by Kim et al. (2021). However, reproducible source code was not available. Therefore, we implemented the Pointer-CRL following the implementation of Bello et al. (2016) [4] and paper of Kim et al. (2021). We set the training step $1,600$ with batch size $B = 100$, which makes a total $160,000$ PI simulations.

Pointer-CIL. Pointer-CIL is an imitation learning version of Pointer-CRL trained by our training data. We set $N = 2000$, $B = 1000$ for training Pointer-CIL.

AM-CRL. AM-CRL is a AM-based DPP solver proposed by Park et al. (2022a). We reproduced AM-CRL by following implementation of Kool et al. (2019)[5] and paper of Park et al. (2022a). We set the training step $2,000$ with batch size $B = 100$, which makes a total of $200,000$ PI simulations.

AM-CIL. AM-CIL is an imitation learning version of AM-CRL trained by our training data. For experiments in Table 1, we set $N = 2000$ and $B = 1000$ for training. For the ablation study, we mainly ablate N, when $N = 100$ we set $B = 100$. Here is the training sample complexity (the number of PI simulations during training) of each ML baseline and DEVFORMER:

Table 4: Training sample complexity of pre-trained ML baselines and DEVFORMER.

Methods	Number of PI simulations for Training
Pointer-CRL	200,000
AM-CRL	200,000
Pointer-CIL $\{N = 2000\}$	200,000 ($N = 2000$, $M = 100$ from GA expert)
AM-CIL $\{N = 2000\}$	200,000 ($N = 2000$, $M = 100$ from GA expert)
DEVFORMER $\{N = 100\}$ (ours)	10,000 ($N = 100$, $M = 100$ from GA expert)
DEVFORMER $\{N = 500\}$ (ours)	50,000 ($N = 500$, $M = 100$ from GA expert)
DEVFORMER $\{N = 1000\}$ (ours)	100,000 ($N = 1000$, $M = 100$ from GA expert)
DEVFORMER $\{N = 2000\}$ (ours)	200,000 ($N = 2000$, $M = 100$ from GA expert)

During the inference phase, each learned model produces a greedy solution from their policies (i.e., $M = 1$) following (Kool et al., 2019).

[4] https://github.com/pemami4911/neural-combinatorial-rl-pytorch
[5] https://github.com/wouterkool/attention-learn-to-route

C.3. Implementation of Meta-Heuristic Baselines.

Genetic Algorithm (GA). GA $\{M = 100\}$ and GA $\{M = 500\}$ are implemented as baselines. For detailed procedures and operators used for GA, see Appendix B. GA $\{M = 100\}$ is the expert policy used to generate expert data for imitation learning in DEVFORMER. For GA $\{M = 100\}$, the size of population, P_0, is 20, number of generation, G, is 5 and elite population, P_{elite}, is 4. For GA $\{M = 500\}$, P_0 is 50, G is 10 and P_{elite} is 10.

Random Search (RS). The random search method generates M random samples for a given problem and selects the best sample with the highest objective value.

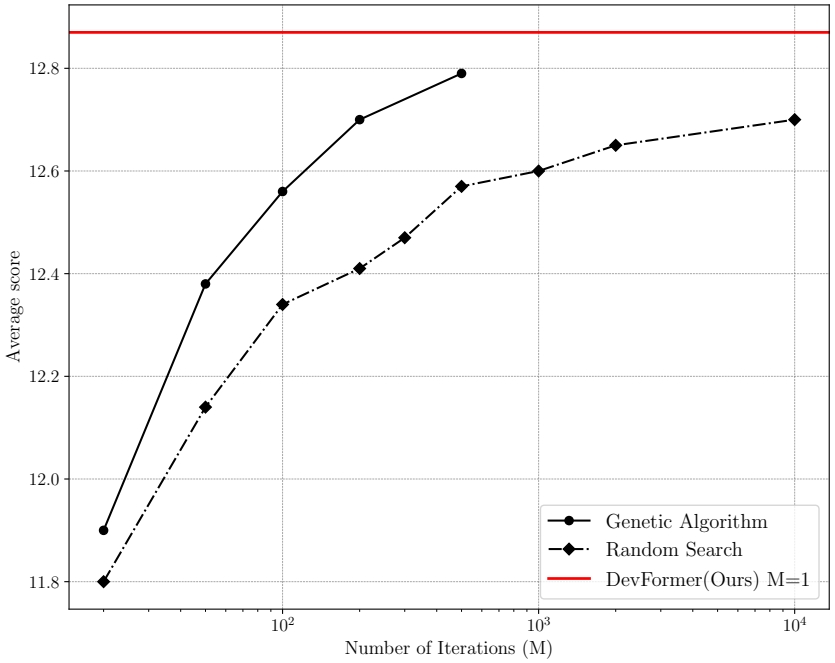


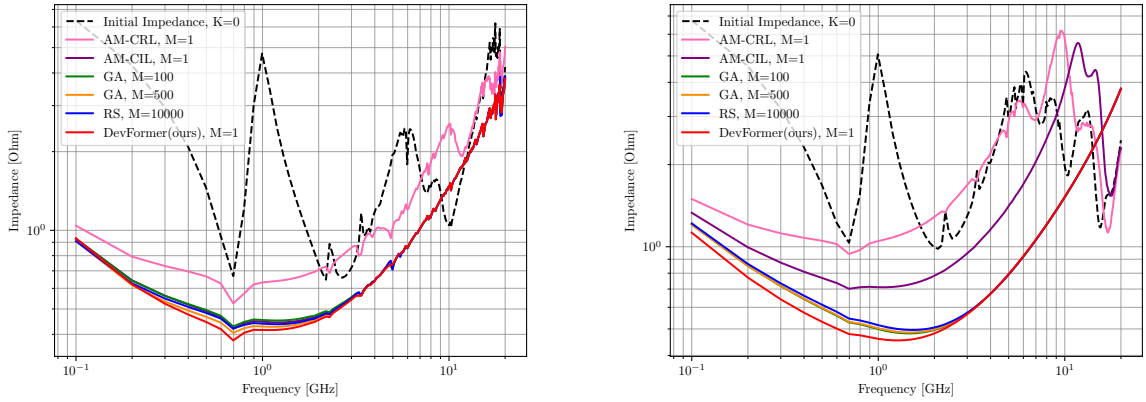
Figure S9: Performance of GA and RS with a varying number of iterations (M) in comparison to DEVFORMER at $M = 1$.

Fig. S9 shows the performance of GA and RS depending on the number of iterations (M). The performance was measured by taking the average of 100 test data solved by each method at each M. GA outperformed RS at every M, and the performance increased with increasing M for both methods. However, the gradient of performance increment decreased with increasing M. On the other hand, our DEVFORMER showed higher performance than GA$\{M = 100\}$ and RS $\{M = 10,000\}$ with a single inference $M = 1$.

D. Experimental Results in terms of Power Integrity

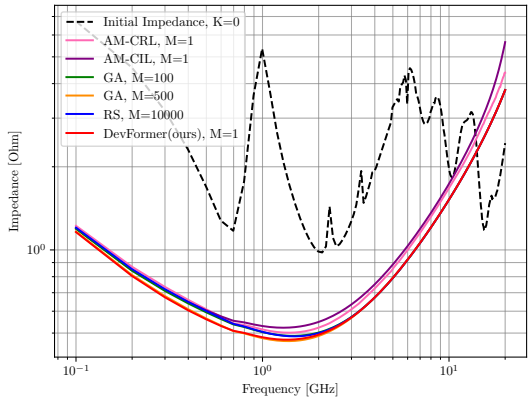
The objective of DPP is to suppress the probing port impedance as much as possible over a specified frequency range and is measured by the objective metric, $Obj := \sum_{f \in F}(Z_{initial}(f) - Z_{final}(f)) \cdot \frac{1\text{GHz}}{f}$. Performance of DEVFORMER was evaluated in comparison to GA $\{M = 100\}$ (*expert policy*), GA $\{M = 500\}$, RS $\{M = 10,000\}$, AM-CRL and AM-CIL on unseen 100 PDN cases. Each method was asked to place 20 decaps ($K = 20$) on each test.

D.1. Impedance Suppression Plots

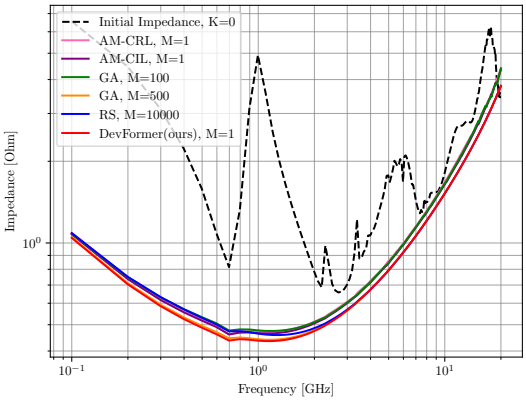


(a) Test Case 1.

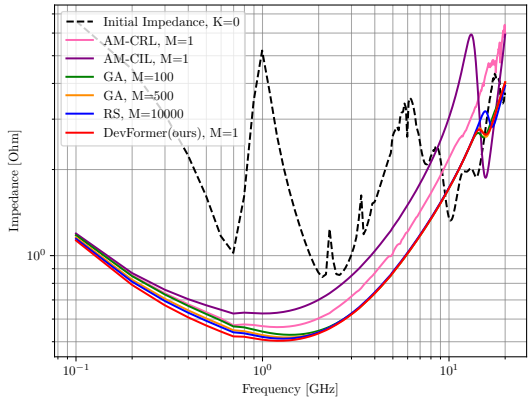
(b) Test Case 2.



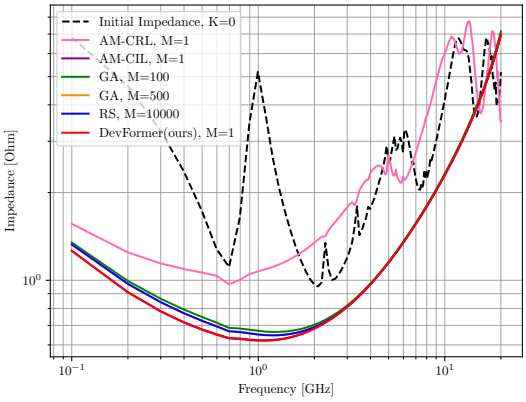
(c) Test Case 3.



(d) Test Case 4.



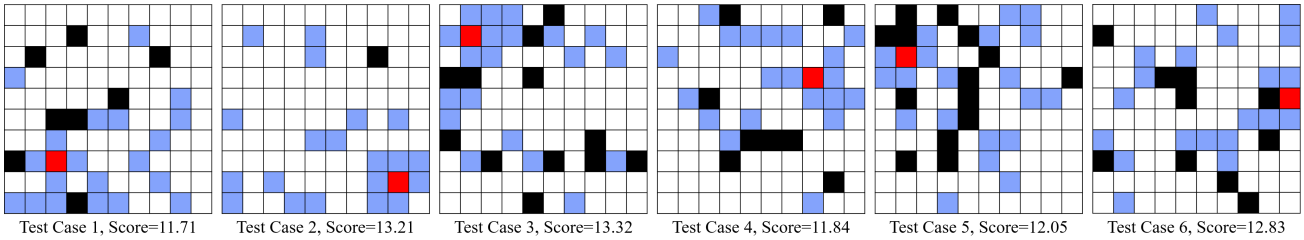
(e) Test Case 5.



(f) Test Case 6.

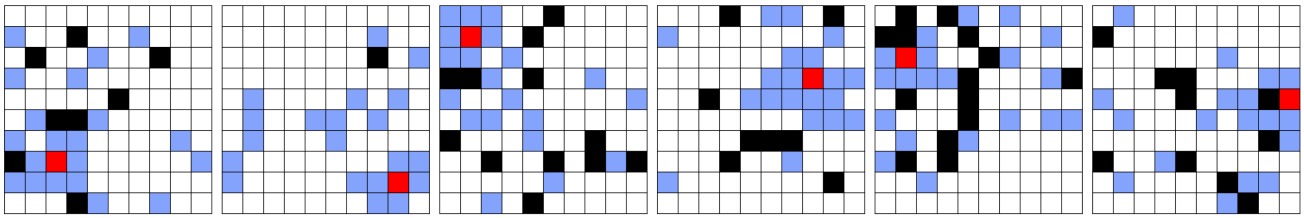
Figure S10: Impedance suppressed by each method, GA $\{M = 100\}$ (*expert policy*), GA $\{M = 500\}$, RS $\{M = 10,000\}$, AM-CRL, AM-CIL and DEVFORMER (Ours) for 6 examples PDN cases out of 100 test dataset. (The lower, the better.)

D.2. Decap Placement Tendency Analysis



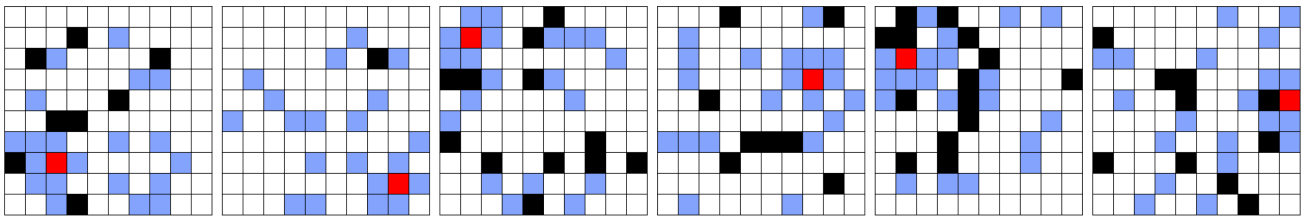
Test Case 1, Score=11.71 Test Case 2, Score=13.21 Test Case 3, Score=13.32 Test Case 4, Score=11.84 Test Case 5, Score=12.05 Test Case 6, Score=12.83

(a) GA $\{M = 100\}$.



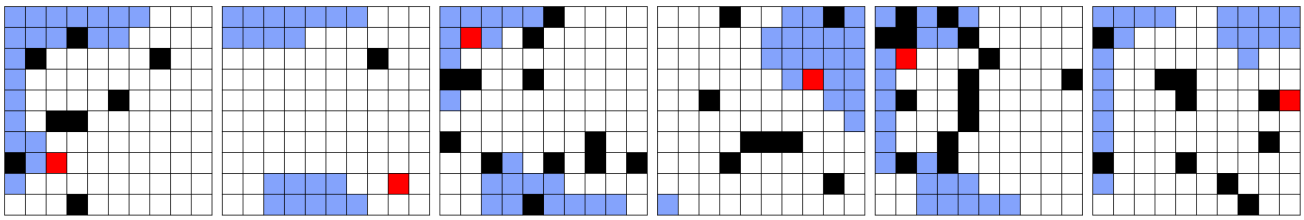
Test Case 1, Score=11.79 Test Case 2, Score=13.20 Test Case 3, Score=13.42 Test Case 4, Score=12.25 Test Case 5, Score=12.13 Test Case 6, Score=12.97

(b) GA $\{M = 500\}$.



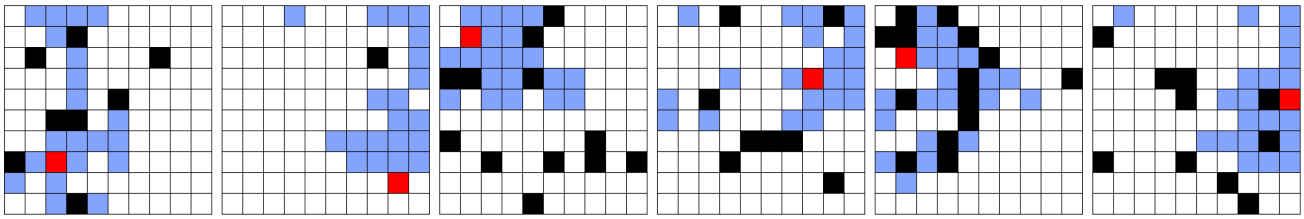
Test Case 1, Score=11.76 Test Case 2, Score=13.15 Test Case 3, Score=13.30 Test Case 4, Score=12.14 Test Case 5, Score=12.12 Test Case 6, Score=12.97

(c) RS $\{M = 10,000\}$.



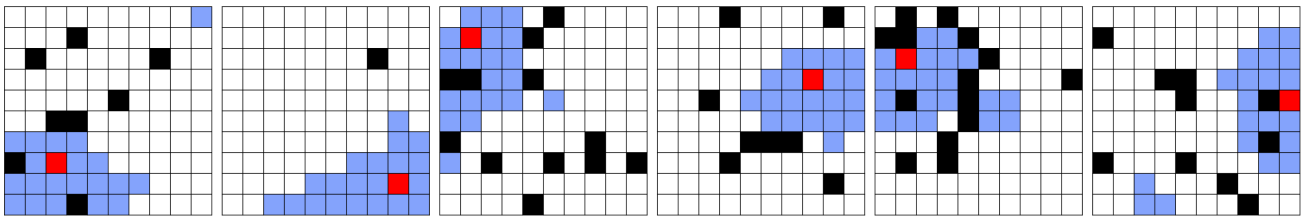
Test Case 1, Score=10.01 Test Case 2, Score=8.51 Test Case 3, Score=12.93 Test Case 4, Score=11.81 Test Case 5, Score=10.97 Test Case 6, Score=8.30

(d) AM-CRL $\{M = 1\}$.



Test Case 1, Score=11.72 Test Case 2, Score=10.58 Test Case 3, Score=12.70 Test Case 4, Score=11.90 Test Case 5, Score=10.31 Test Case 6, Score=13.14

(e) AM-CIL $\{M = 1\}$.



Test Case 1, Score=11.80 Test Case 2, Score=13.43 Test Case 3, Score=13.43 Test Case 4, Score=12.28 Test Case 5, Score=12.22 Test Case 6, Score=13.15

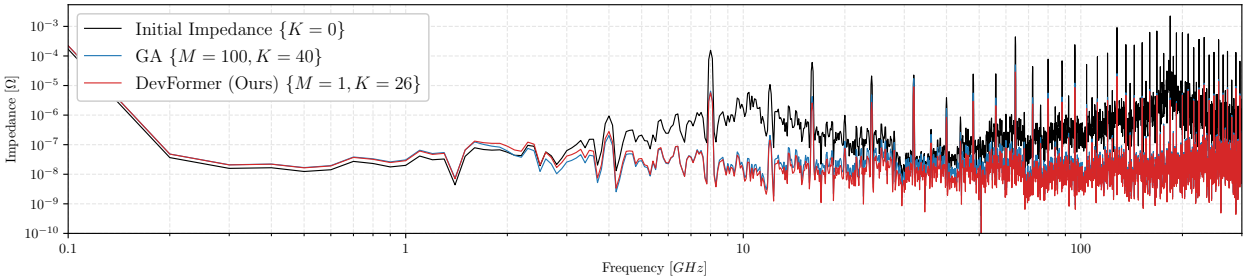
(f) DEVFORMER(ours)$\{M = 1\}$.

Figure S11: Corresponding decap placement solutions to Fig. S10 by each method. Red represents probing port, black represents keep-out ports and blue represents decap locations.

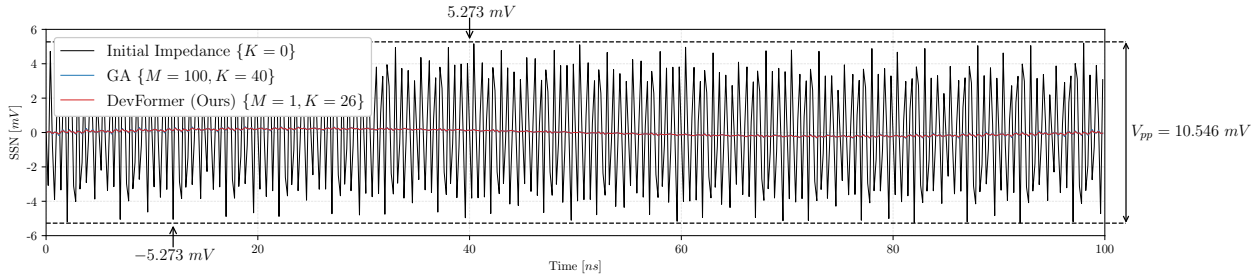
Fig. S12 shows the decap placement solutions of 6 PDN cases plotted in Fig. S10. The solutions by the search-heuristic methods, GA and RS, tend to be scattered, while the solutions by learning-based methods, AM-CRL, AM-CIL, and DEVFORMER, are clustered. Since search-heuristic methods are based on random generations, they do not show a clear tendency. On the other hand, learning-based methods are based on a policy, having a distinct tendency to place decaps.

The role of placing decaps in hardware design is to decouple the loop inductance of PDN. In terms of PI, analysis of loop inductance is critical, but at the same time, is complex (Farrahi & Koether, 2019). The loop inductance distribution of PDN highly depends on various design parameters such as the location of the probing port, spacing between power/ground, size of PDN, and hierarchical layout of PDN (Fan et al., 2000). When human experts place decaps on PDN, there are too many domain rules to consider. On the other hand, DEVFORMER understands the PDN structure and its electrical properties by data-driven learning. According to Fig. S12, DEVFORMER tends to place decaps near the probing port, which is a well-known expert rule in the PI domain.

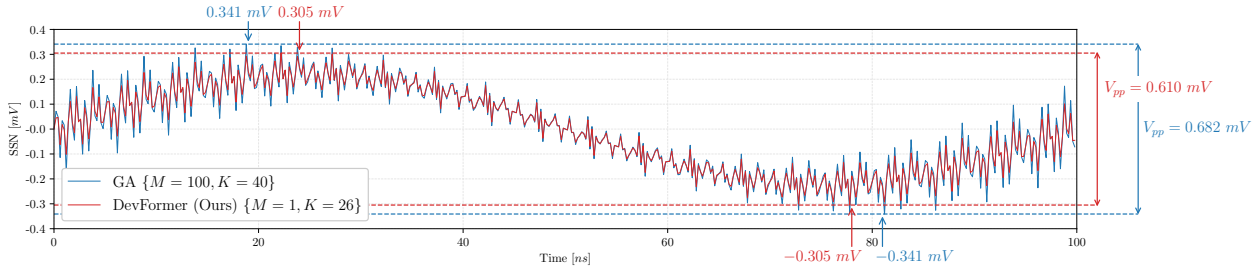
D.3. Power Noise Analysis on HBM PDN



(a) Impedance suppression



(b) Initial power noise before decap placement



(c) Power noise after decap placement by our DEVFORMER$\{M = 1, K = 26\}$ and GA$\{M = 100, K = 40\}$

Figure S12: Power noise analysis in terms of simultaneous switching noise (SSN) on HBM PDN before and after decap placement by our DEVFORMER$\{M = 1, K = 26\}$ and GA$\{M = 100, K = 40\}$. DEVFORMER reduces power noise more than GA while reducing the needed decap number K by more than 30%.

Appendix D.3 analyzes the performance of DEVFORMER in comparison to GA$\{M = 100\}$ in terms of power noise. Out of 100 test cases on HBM PDN, we randomly chose a test case and carried out peak-to-peak power noise analysis for a circuit block, phase-locked loop (PLL), operating at 5GHz. Note that DEVFORMER placed 26 decaps and GA$\{M = 100\}$ placed 40 decaps. DEVFORMER reduced power noise more than GA$\{M = 100\}$ with 14 less decaps. The impedances of the probing port on the power distribution network (PDN) before and after decap placed by DEVFORMER and GA$\{M = 100\}$ are presented in Fig. S12a. The time-domain power noise before and after decap placement by each method is shown in Fig. S12b. For performance comparison, Fig. S12c shows the time-domain power noise after decap placement by DEVFORMER and GA$\{M = 100\}$.

E. Further Ablation Study

This section reports further ablation studies on the hyperparameters N (number of offline expert data used), λ (weight of self-exploitation loss term), and P (number of permutation transformed labels).

E.1. Ablation Study on N

N is the number of expert labels generated by the expert policy, GA $\{M = 100\}$. We ablate $N \in \{100, 500, 1000, 2000\}$ with fixed $P = 4$ and $\lambda = 5$ and compare to AM-CIL baseline for all N. As shown in Table 5, DEVFORMER with $N = 2000$ gives the best performance, and DEVFORMER outperforms AM-CIL for all N variations. The performance of AM-CIL is saturated at $N > 500$ while the performance of DEVFORMER continuously increases with the increase of N.

Table 5: Ablation study on N for DEVFORMER ($P = 4, \lambda = 8$) and AM-CIL.

	Validation Score
AM-CIL $\{N = 100\}$	11.02
DEVFORMER (ours) $\{N = 100\}$	**12.76**
AM-CIL $\{N = 500\}$	11.80
DEVFORMER (ours) $\{N = 500\}$	**12.85**
AM-CIL $\{N = 1000\}$	11.99
DEVFORMER (ours) $\{N = 1000\}$	**12.86**
AM-CIL $\{N = 2000\}$	11.77
DEVFORMER (ours) $\{N = 2000\}$	**12.88**

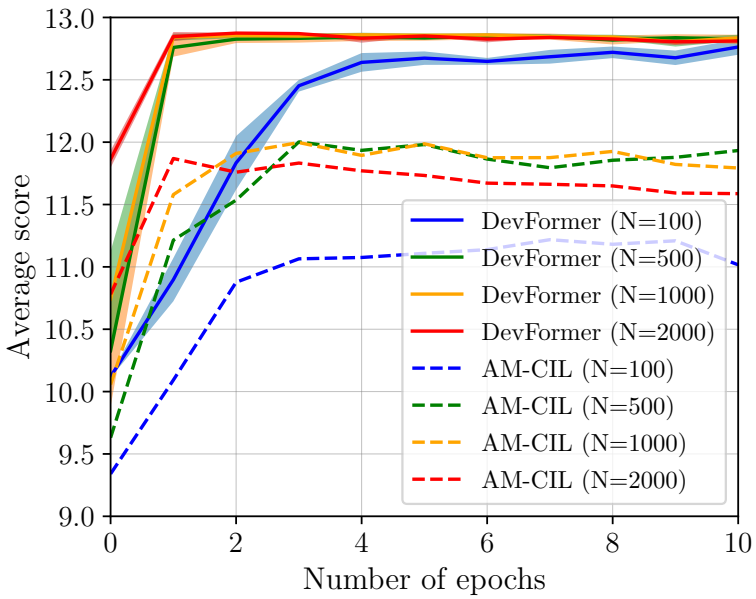


Figure S13: Validation graph of DEVFORMER and AM-CIL for varying number of offline expert data $N \in \{100, 500, 1000, 2000\}$.

E.2. Ablation Study on λ

λ refers to the weight of self-exploitation loss term L_{Self}, in the collaborative learning loss $\mathcal{L} := \mathcal{L}_{Expert} + \lambda\mathcal{L}_{Self}$. To set $\lambda \times L_U$ to be $0.1 \sim 1$, we first multiplied 10^{32} to λ because the probability of a specific solution is extremely small. Then, we ablated for $\lambda \in \{1, 2, 4, 6, 7, 8, 9, 10\}$ (10^{32} is omitted) with fixed $N = 2000$ and $P = 4$. For every λ, it prevents overfitting of the model in comparison to the baselines trained only with L_{Expert} (see Fig. S14). According to the Table 7, $\lambda = 8$ gives the best validation scores.

Table 6: Ablation study of λ on fixed $P = 4$ and $N = 2000$.

λ ($\times 10^{32}$)	Validation Score
1	12.863
2	12.865
3	12.866
4	12.870
5	**12.877**
6	12.874
7	12.862
8	12.871
9	12.871
10	12.870
Only IL	11.832

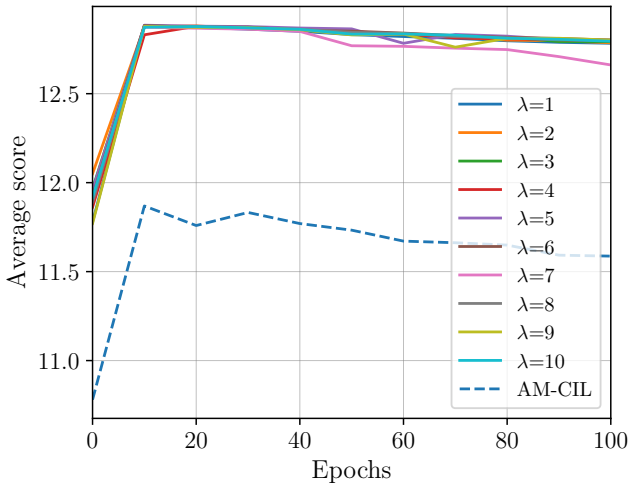


Figure S14: Validation graph of $\lambda \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ on fixed $P = 4$ and $N = 2000$.

E.3. Ablation Study on P

P is the number of permutation-transformed labels per each expert label used for imitation learning-based expert exploitation. We ablate $P \in \{4, 6, 8\}$ with fixed $N = 2000$ and $\lambda = 5$ and compared collaborative symmetricity exploitation (i.e., both expert and self-exploitation) to only expert exploitation training cases. As shown in Table 7, $P = 4$ with {Expert exploitation + Self-exploitation} give best performances. For every P, {Expert exploitation + Self-exploitation} gives better performances, indicating the self-exploitation scheme well prevents overfitting the training process for the sparse dataset.

Table 7: Ablation study on P with and without unsupervised loss term.

	Validation Score
Expert exploitation $\{P = 4\}$	12.85
+ Self- exploitation $\{\lambda = 5\}$	**12.88**
Expert exploitation $\{P = 6\}$	12.87
+ Self- exploitation $\{\lambda = 5\}$	**12.88**
Expert exploitation $\{P = 8\}$	12.88
+ Self- exploitation $\{\lambda = 5\}$	**12.88**

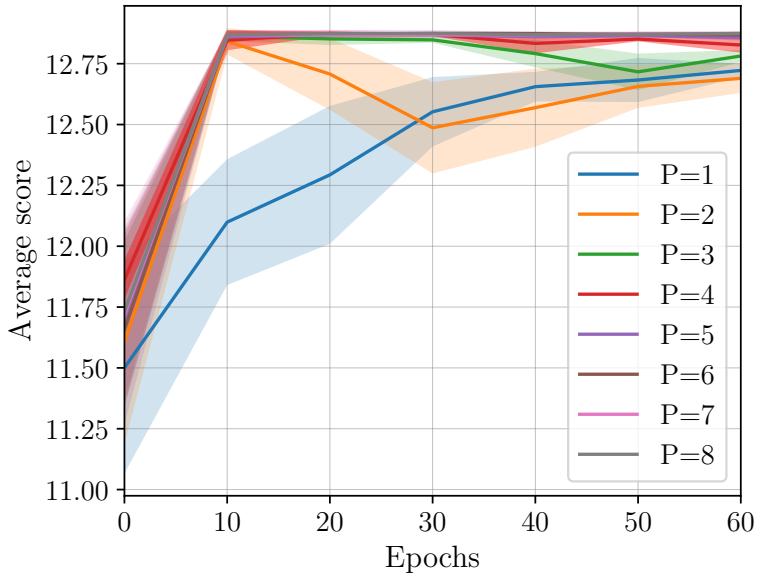


Figure S15: Validation score of P ablation with and without self-exploitation loss term.

F. Proof of Theorem 1

($\rightarrow$) Suppose that policy $\pi(\boldsymbol{a}|\boldsymbol{x})$ is AP-symmetric. Then, by the Definition 1, $\pi(\boldsymbol{a}|\boldsymbol{x}) = \pi(t(\boldsymbol{a})|\boldsymbol{x})$ for any $\boldsymbol{a} \in \mathcal{A}$, $\boldsymbol{x} \in \mathcal{X}$, $t \in T_{AP}$.

Therefore,

$$b(\pi; \boldsymbol{p}) = \mathbb{E}_{\boldsymbol{x} \sim p_{\mathcal{X}}(x)} \mathbb{E}_{\boldsymbol{a} \sim p_{\mathcal{A}}(\boldsymbol{a})} \mathbb{E}_{t \sim p_{T_{AP}}(t)} [||\pi(\boldsymbol{a}|\boldsymbol{x}) - \pi(t(\boldsymbol{a})|\boldsymbol{x})||_1] = 0$$

($\leftarrow$) Suppose that $b(\pi; \boldsymbol{p}) = 0$, where $p_{\mathcal{X}}(\boldsymbol{x}) > 0, p_{\mathcal{A}}(\boldsymbol{a}) > 0, p_{T_{AP}}(t) > 0$.

Assume that there exist $\boldsymbol{a^*} \in \mathcal{A}$, $\boldsymbol{x^*} \in \mathcal{X}$, and $t^* \in T_{AP}$, such that $\pi(\boldsymbol{a}^*|\boldsymbol{x}^*) \neq \pi(t(\boldsymbol{a}^*)|\boldsymbol{x}^*)$.

Then,

$$\begin{aligned} b(\pi; \boldsymbol{p}) &= \mathbb{E}_{\boldsymbol{x} \sim p_{\mathcal{X}}(x)} \mathbb{E}_{\boldsymbol{a} \sim p_{\mathcal{A}}(\boldsymbol{a})} \mathbb{E}_{t \sim p_{T_{AP}}(t)} [||\pi(\boldsymbol{a}|\boldsymbol{x}) - \pi(t(\boldsymbol{a})|\boldsymbol{x})||_1] \\ &\geq p_{\mathcal{X}}(\boldsymbol{x}^*) p_{\mathcal{A}}(\boldsymbol{a}^*) p_{T_{AP}}(t^*) ||\pi(\boldsymbol{a}^*|\boldsymbol{x}^*) - \pi(t(\boldsymbol{a}^*)|\boldsymbol{x}^*)||_1 > 0, \end{aligned}$$

which results in a contradiction. Therefore, $\pi(\boldsymbol{a}|\boldsymbol{x}) = \pi(t(\boldsymbol{a})|\boldsymbol{x})$ for any F$\boldsymbol{a} \in \mathcal{A}$, $\boldsymbol{x} \in \mathcal{X}$, $t \in T_{AP}$: i.e, policy $\pi(\boldsymbol{a}|\boldsymbol{x})$ is AP-symmetric.