COLLABORATIVE SYMMETRICITY EXPLOITATION FOR OFFLINE LEARNING OF HARDWARE DESIGN SOLVER

Anonymous authors
Paper under double-blind review

ABSTRACT

This paper proposes *collaborative symmetricity exploitation* (CSE) framework to train a solver for the decoupling capacitor placement problem (DPP) benchmark, one of the significant hardware design problems. Due to the sequentially coupled multi-level property of the hardware design process, the design condition of DPP changes depending on the design of higher-level problems. Also, the online evaluation of real-world electrical performance through simulation is extremely costly. Thus, a data-efficient offline learning method to train a solver (i.e., contextualized policy) with high generalization capability over changing task conditions is necessary. In this paper, we apply the CSE framework to train a DPP solver using a limited number of offline expert data. Leveraging the symmetricity for offline learning of hardware design solver has two major advantages: it increases data-efficiency by reducing the solution space and improves generalization capability by capturing the invariant nature present regardless of changing conditions. The proposed CSE is composed of two learning schemes: expert exploitation and self-exploitation. Expert exploitation induces symmetricity during the imitation learning process with offline expert data and self-exploitation induces symmetricity during the consistency learning process with self-generated data. Extensive experiments verified that CSE with zero-shot inference outperforms the neural baselines and iterative conventional design methods on the DPP benchmark. Furthermore, CSE showed promising extrapolation capability as it greatly outperforms the expert method used to generate the offline data for training. Scalability and flexibility of the proposed method were also verified for practical use of CSE in industry.

1 INTRODUCTION

With the CMOS technology shrinking and increasing data rate, the design complexity of very large-scale integrated (VLSI) has increased. Human experts are no longer able to design hardware without the help of electrical design automation (EDA) tools, and EDA tools now suffer from long simulation time and insufficient computing power, making machine learning (ML) application to hardware design inevitable. Many studies have already shown that deep reinforcement learning (DRL), one of the representative ML methods for sequential decision making, is promising in various tasks in modern chip design; 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, most previous DRL-based hardware design methods do not take the following into consideration. (a) Online simulators for hardware are usually time intensive and inaccurate; thus, learning with existing offline data by experts is more reliable. Since there exists a limited number of offline hardware data, a data-efficient learning scheme is necessary. (b) Hardware design is composed of electrically coupled multi-level tasks where task conditions are determined by the design of higher-level tasks; thus, a solver (i.e., contextualized policy conditioned by higher-level tasks) with high generalization capability to adapt to varying task conditions is necessary.

We observed that leveraging the symmetricity in placement problems can effectively improve both data-efficiency of training and generalization capability of trained solver over task condition variation. As shown in Fig. 1, the conventional sequential decision-making schemes for a placement problem (Park et al., 2020; Mirhoseini et al., 2021; Cheng & Yan, 2021) search over a heterogeneous trajectory,

referring to a group of $K!$ action-permutation (AP)-symmetric solution trajectories where K is the number of actions. If each group of AP-symmetric solution trajectories can be identified as a single merged trajectory, the search space can be dramatically reduced, giving two major advantages to tackling the target hardware problem. First, inducing symmetricity allows data-efficient training of solver with a limited number of offline expert data because the amount of training data is proportional to the policy's search space. Second, inducing symmetricity improves generalization capability of the trained solver to varying task conditions because symmetricity is an invariant nature shared among tasks regardless of changing conditions.

To this end, we devised *collaborative symmetricity exploitation* (CSE) framework, a simple but effective method to induce AP-equivariance with two collaborative learning schemes: expert exploitation and self-exploitation. The expert exploitation simply augments the offline expert data (sequential data) with a random permutation operator and uses it for imitation learning. The self-exploitation generates pseudo-labeled solutions from the current training policy, transforms the pseudo-labeled solution with a random permutation operator, and forces the solver to have an identical probability to generate the original pseudo-labeled solution and the transformed solution.

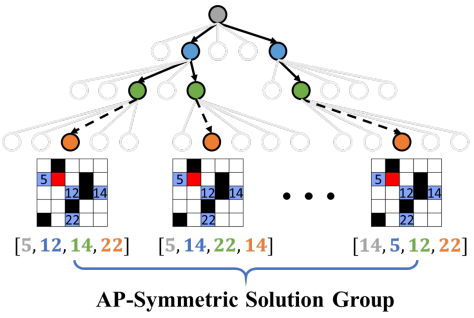


Figure 1: Conventional sequential decision-making method's heterogeneous trajectories from AP-Equvariant solution group.

To verify the effectiveness of CSE, we applied CSE to the decoupling capacitance (decap) placement problem (DPP), one of the significant hardware design benchmarks. The objective of DPP is to place a given number of decaps on power distribution network (PDN) with two varying conditions: keep-out regions and probing port location, determined by higher-level problems such as chip placement and routing. The goal of CSE is to train a solver (i.e., contextualized policy) that has high generalization capability to any given task condition.

To further improve the generalization capability of the trained solver, we also devised a DPP-specific neural architecture by modifying the attention model (AM) (Kool et al., 2019) with two problem-specific context neural networks, the probing port context network (PCN) and the recurrent context network (RCN). PCN captures the location of the probing port, which is a major context of DPP tasks. RCN recurrently captures the current state to support the sequential decoding scheme. See Fig. 2 for the overall pipeline of solving DPP.

We expect the proposed CSE will have the following industrial impacts:

- **Adaptable solver generation**: The DPP solver trained by CSE is scalable to the action and state spaces of given hardware tasks (Table 2). Furthermore, the CSE learning framework is flexible to be applied to various hardware targets (Section 5.5).
- **Simulation cost reduction**: The DPP solver with high generalization capability can be trained only with a limited number of existing offline data by CSE (Section 5.3).
- **Fabrication cost reduction**: The DPP solver trained by CSE can reduce the number of decaps while satisfying the target. Since hardware devices are mass-produced, reducing a single decap can enormously reduce hardware fabrication costs. (Section 5.4 and Section 5.5).
- **Power noise reduction**: The DPP solver trained by CSE can effectively reduce the power noise in real-world hardware devices, which is crucial for high-speed high performance systems (Section 5.5 and Appendix E.3).

2 Decap Placement Problem (DPP) Formulation

This paper seeks to solve the decoupling capacitor placement problem (DPP), one of the essential hardware design problems. Decoupling capacitor (decap) is a hardware component that reduces power noise along with the power distribution network (PDN) of hardware devices and improves the power integrity (PI). With transistor scaling and continuously decreasing supply voltage margin (Hwang et al., 2021), power noise has become a huge technical bottleneck in high-speed computing

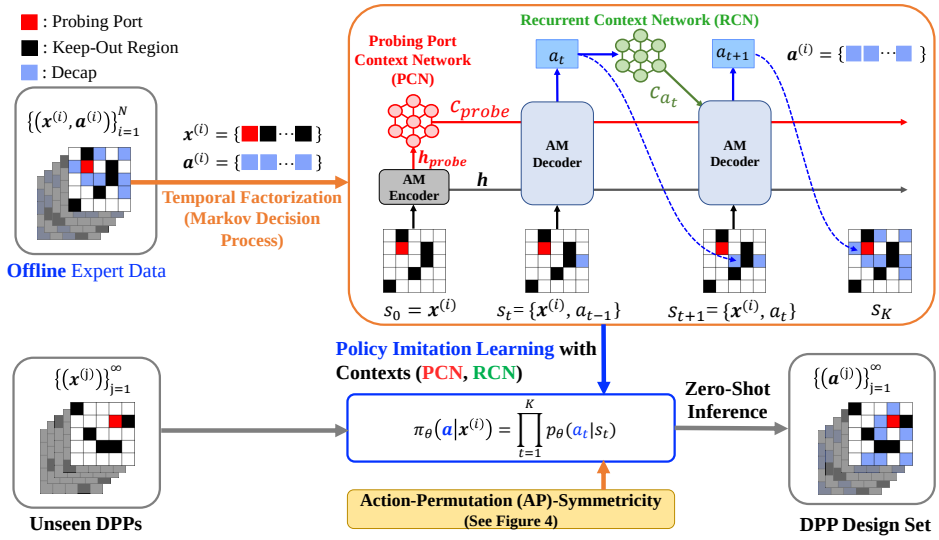


Figure 2: Overall pipeline of DPP contextualized policy parameterization process with offline learning process.

systems. Generally, the more decaps are placed, the more reliable the power supply is. However, adding more decaps requires more space and is costly. Thus, finding an optimal placement of decaps is essential in terms of hardware performance and cost/space-saving.

The goal of DPP benchmark is to optimally place a pre-defined number of decaps on a target PDN, given two conditions determined by higher-level tasks; keep-out regions and a probing port location (Kim et al., 2021). Keep-out regions are action-restricted areas where decaps cannot be placed as a design constraint. Probing port is the target chip/logic block location where the objective, power integrity (PI), is evaluated. The ports on PDN are represented as $N_{row} \times N_{col}$ grids, and the number of decaps is denoted by K. See Appendix A.2 for details of PDN and decap modeling for the benchmark.

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. This study aims to learn an effective DPP solver that can be used in practice.

2.1 Markov Decision Process (MDP)

As shown in Fig. 2, the procedure for solving DPP is modeled as a Markov decision process(MDP). The *task-condtioned* PDN is represented as a set of three-dimensional feature vectors $\boldsymbol{x} = \{\boldsymbol{x}_i\}_{i=1}^{N_{row} \times N_{col}}$, where each grid (i.e., port) on PDN is represented as $\boldsymbol{x}_i = (x_i, y_i, c_i)$, in which x_i, y_i indicate 2D coordinates of location, c_i indicates the condition of port whether it belongs to a probing port I_{probe} ($c_i = 2$), keep-out regions $I_{keepout}$ ($c_i = 1$), or decap allowed ports $I_{allowed}$ ($c_i = 0$). Note that $I_{keepout}$ and $I_{allowed}$ represent index sets consisting of indices corresponding to keep-out regions and decap allowed ports, respectively. I_{probe} refers to an index of probing port. See Appendix A.3.

The design process sequentially places decaps on the available PDN ports until planning all the designated K decaps. We model this sequential Markov decision process with state, action, and policy as follows:

State $\boldsymbol{s}_t$ contains task-condition $\boldsymbol{x}$ and previous selected actions: $\boldsymbol{s}_t = \{\boldsymbol{x}, \boldsymbol{a}_{1:t-1}\}$.

Action $a_t \in \{1, ..., N_{row} \times N_{col}\} \setminus \boldsymbol{s}_{t-1}$ 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 ports, and the 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 $\boldsymbol{a} = \boldsymbol{a}_{1:K}$, given task-condition $\boldsymbol{x}$, 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.

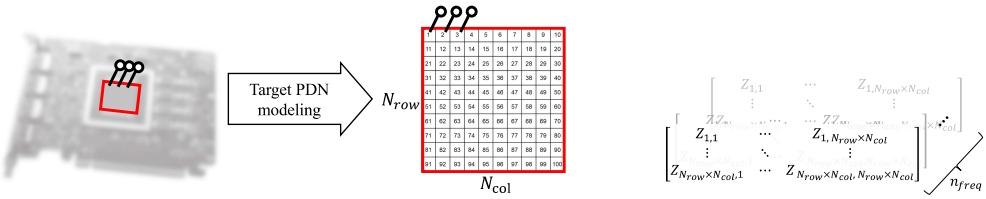
The objective of DPP is to find the optimal parameter θ^* of the policy $\pi_\theta(\cdot|\boldsymbol{x})$ as:

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

where ρ is the probability distribution for varying *task-condition* $\boldsymbol{x}$ and $\mathcal{J}$ is objective function. Finding the optimal policy for various DPPs is a task contextual learning problem, in which each DPP has a distinct task condition. Once the task $\boldsymbol{x}$ is specified by ρ, the state-action space with complexity of $\binom{N_{row}\times N_{col}-1-|I_{keepout}|}{K}$ is determined. Thus, an efficient policy $\pi_\theta(\boldsymbol{a}|\boldsymbol{x})$ should capture the contextual features among varying task conditions $\boldsymbol{x}$.

Note that CSE is based on imitation learning, by which objective function $\mathcal{J}$ is implicitly induced through offline expert data. The objective function of DPP is described in equation 3.

2.2 Objective Function of DPP



(a) Unit-cell representation of target PDN of a hardware device

(b) Z-parameter of target PDN.

Figure 3: Unit-cell and Z-parameter representations of real-world target PDN.

Performance Evaluation Metric. The performance of DPP is evaluated by power integrity (PI) simulation that computes the level of impedance suppression over a specified frequency domain and is quantified as:

$$\mathcal{J} := \sum_{f\in F}(Z_{initial}(f) - Z_{final}(f)) \cdot \frac{1\text{GHz}}{f} \tag{3}$$

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. As shown in Fig. 3, the PI simulation for $(N_{row} \times N_{col})$ PDN requires a $N_{row}N_{col} \times N_{row}N_{col} \times n_{freq}$ (number of frequency points) sized Z-parameter matrix calculation because each port is electrically coupled to the rest of the ports and Z (i.e., impedance) is frequency-dependent. Thus, performance evaluation with a large Z-parameter matrix calculation is costly. The more impedance is suppressed, the better the power integrity and the higher the performance score. Note that this performance metric was also used for collecting the offline expert data using a genetic algorithm (GA).

3 Methodology

This section provides technical details of the proposed *collaborative symmetricity exploitation* (CSE) framework and the modified attention model (AM) neural architecture with two domain-specifically devised context neural networks for training a DPP solver.

3.1 Action-permutation Symmetricity of Placement Task

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

Definition 3.1 (**AP-symmetricity**). For any $\boldsymbol{a} \in \mathcal{A}$, $\boldsymbol{x} \in \mathcal{X}$, $t \in T_{AP}$ where $\mathcal{A}$ is solution space and $\mathcal{X}$ is task-condition space,

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

The objective function $\mathcal{J} : \mathcal{X} \times \mathcal{A} \rightarrow \mathbb{R}$ is an AP-symmetric function because $\boldsymbol{a}$ and $t(\boldsymbol{a})$ have identical placement design. The main role of CSE is to induce AP-symmetricity to the policy π (conditional probability) to reflect the AP-symmetricity of an objective function $\mathcal{J}$.

3.2 Collaborative Symmetricity Exploitation (CSE) Framework

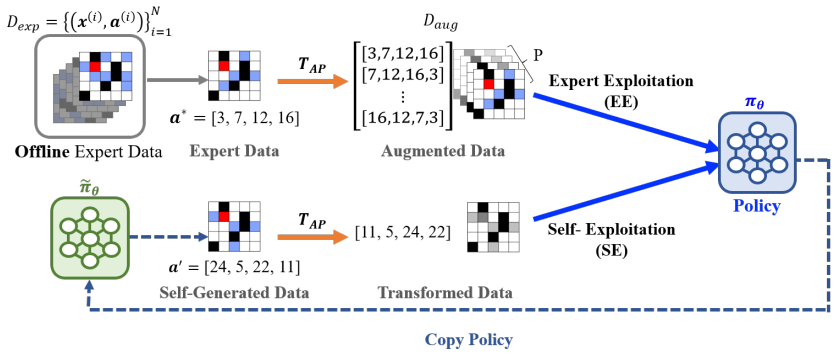


Figure 4: Illustration of collaborative symmetricity exploitation (CSE) process.

The CSE framework was designed to induce the AP-symmetricity to the trained model to improve the generalization capability and to allow data-efficient training.

To train a contextualized policy with a limited number of expert data, we designed the CSE loss term $\mathcal{L}$ consisting of expert exploitation loss $\mathcal{L}_{Expert}$ and self-exploitation loss $\mathcal{L}_{Self}$ as follows:

$$\mathcal{L} := \mathcal{L}_{Expert} + \lambda \mathcal{L}_{Self} \tag{4}$$

$$\mathcal{L}_{Expert} = -\mathbb{E}_{\boldsymbol{a}^*, \boldsymbol{x} \sim D_{aug}}[log \pi_\theta(\boldsymbol{a}^*|\boldsymbol{x})] \tag{5}$$

$$\mathcal{L}_{Self} = \mathbb{E}_{\boldsymbol{x} \sim \mathcal{U}(\mathcal{X})} \mathbb{E}_{\boldsymbol{a}' \sim \pi_{\tilde{\theta}}(\cdot|\boldsymbol{x})} \mathbb{E}_{t \sim \mathcal{U}(T_{AP})}[||\pi_{\tilde{\theta}}(\boldsymbol{a}'|\boldsymbol{x}) - \pi_\theta(t(\boldsymbol{a}')|\boldsymbol{x})||_1] \tag{6}$$

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

Self-Exploitation. While D_{aug} only contains expert quality, self-exploitation involves self-generated data, whose quality is poor at the beginning but improves over the phase of training. Thus, the self-exploitation scheme is designed to induce the AP-symmetricity in a wider action space to achieve greater generalization capability. The proposed $\mathcal{L}_{Self}$ has three probability distributions for expectation: $\mathcal{U}(\mathcal{X})$, $\pi_{\tilde{\theta}}$, and $\mathcal{U}(T_{AP})$. First, the $\mathcal{U}(\mathcal{X})$ indicates a uniform distribution of task-condition set $\mathcal{X}$ which helps to learn a contextualized policy capable of adaptation to task variations. Second, the $\pi_{\tilde{\theta}}$ is a fixed copy of the current policy during the training, which samples a pseudo label data $\boldsymbol{a}'$ for task-condition $\boldsymbol{x}$. Lastly, the $\mathcal{U}(T_{AP})$ is a uniform distribution that samples permutation t from T_{AP}.

With the three probability distributions, $\mathcal{L}_{Self}$ is expressed as an expectation of L_1 loss between two probabilities: the probability to generate $\boldsymbol{a}'$ and the probability to generate $t(\boldsymbol{a}')$. By minimizing L_{self}, we can enforce the probabilities of generating a' and $t(a')$ to be identical, thus imposing AP-symmetricity directly into the current policy π.

3.3 CONTEXTUAL ATTENTION MODEL

To further improve the generalization capability of the trained DPP solver, we modified the attention model (AM) (Kool et al., 2019) and termed *contextual attention model*. As described in Fig. 2, the decision-making procedure consists of two newly devised context neural networks; (1) encoder capturing initial design conditions while contextualizing the probing port through the probing port context network (PCN), and (2) decoder sequentially allocating decaps on PDN while contextualizing the stages of the partial solution through the recurrent context network (RCN). See Appendix C for detailed implementation.

Encoder. Our encoder consists of multi-head attention (MHA) and feedforward (FF), similar to the transformer network (Vaswani et al., 2017). The encoder takes the task-conditioned PDN feature vector $\boldsymbol{x}$ as an input and outputs all node embedding $\boldsymbol{h}$. Encoding is processed once at the initial state, $t = 0$. The node embedding $\boldsymbol{h}$ is time-invariant (i.e., fixed after the encoding process) and is used in the decoding process. We proposed a novel probing context network (PCN) in the encoding process so that the learned solver can adapt well to a new task. PCN is a simple but effective two-layer perceptron model with a ReLU activation layer that takes the hidden embedding of the probing port node $\boldsymbol{h}_{probe}$ and outputs the probing port contextual vector $\boldsymbol{c_{probe}} = \mathbf{MLP}_{PCN}(\boldsymbol{h_{probe}})$.

Decoder. With the node embedding $\boldsymbol{h}$ generated by the encoder, decoder sequentially selects an action a_t until placing all K decaps. At each step t, the decoder takes (1) the node embedding $\boldsymbol{h}$ (static information), (2) the current state $\boldsymbol{s_t}$, (3) the probing port contextual vector $\boldsymbol{c_{probe}}$, and (4) the previous action context vector $\boldsymbol{c_{a_{t-1}}} = \mathbf{MLP}_{RCN}(\boldsymbol{h_{a_{t-1}}})$ generated by the recurrent context network (RCN) as inputs and outputs a new action a_t. To leverage sequential state transitions in decoder, we devised the recurrent context network (RCN). RCN is a two-layer perceptron model with a ReLU activation layer that embeds the previously selected node's embedding $\boldsymbol{h}_{a_{t-1}}$ at step t into $\boldsymbol{c_{a_{t-1}}}$. Then, overall context vector is updated as $\boldsymbol{c} = \boldsymbol{c_{probe}} + \boldsymbol{c_{a_{t-1}}}$ and is used as query for attention decoder that eventually infers the next action a_t by the attention mechanism, where the key and value comes from $\boldsymbol{h}$. See Appendix C.2 for detailed process.

4 RELATED WORKS

Machine Learning-based Methods for DPP. Deep reinforcement learning (DRL) has been widely used to solve DPP. Park et al. (2018) employed Q-learning and Park et al. (2020); Zhang et al. (2020) applied convolutional neural network (CNN)-based Q approximators to solve a target DPP. However, their methods require large iterations involving costly reward calculations and their trained policies were non-reusable; if the DPP condition changes, they must be re-trained. In an effort to overcome the reusability limitation by training a solver, Park et al. (2022); Kim et al. (2021) implemented promising neural combinatorial optimization (NCO) models, the attention model (Kool et al., 2019) and pointer network (Vinyals et al., 2015), to construct a contextualized policy without iterative exploration and domain knowledge. However, their methods still showed poor data-efficiency in training and unsatisfactory generalization performance.

Symmetricity Learning in Solution Space. There exists several studies to leverage the symmetricity in solution space. Kwon et al. (2020) suggested a new reinforcement learning scheme, a policy optimization for multiple optima (POMO) to leverage the traveling salesman problem (TSP)'s solution symmetricity, the cyclic property that identical solution can be expressed as N heterogeneous trajectories by permuting initially visited node. Kim et al. (2022) proposed the symmetric neural combinatorial optimization (Sym-NCO) method, which is an extension of POMO to general-purpose symmetric learning for various combinatorial optimization tasks. Bengio et al. (2021) proposed a generative flow net (GFlowNet) to train policy distribution proportional to reward distribution $\pi \propto R$ considering solution symmetricity. The GFlownet suggests a sequential decision-making scheme with a directed acyclic graph (DAG), instead of classical tree structure, to induce solution symmetricity. The GFlowNet is applied to solve molecule optimization and bio-sequential design (Jain et al., 2022).

While POMO (Kwon et al., 2020) and Sym-NCO Kim et al. (2022) leverage DRL, CSE focuses on offline imitation learning. Though GFlowNet (Bengio et al., 2021) can be trained in a fully offline manner, it is not yet designed for training a contextualized policy. Thus, CSE is positioned between POMO and Gflownet as an offline symmetricity learning method to train contextualized policy.

Table 1: Performance evaluation with the average score of 100 PDN cases (the higher the better).

Method	Method Type	PI Simulation (M)	Avg. Score
Random Search	Online Search	10,000	12.70
Genetic Algorithm (*expert policy*)	Online Search	100	12.56
Genetic Algorithm	Online Search	500	12.79
AM-RL (Park et al., 2022)	Pretrained	1	11.71
Arb-RL (Kim et al., 2021)	Pretrained	1	9.60
AM (Park et al., 2022)-IL	Pretrained	1	12.06
Arb (Kim et al., 2021)-IL	Pretrained	1	10.80
CSE (*ours*)	Pretrained	1	**12.88**

5 Experimental Results

5.1 Setup

PDN and Decap Specifications. The PDN used for verification is a chip-package hierarchical PDN, modeled by the segmentation method (Kim et al., 2010; Cho et al., 2019). The PDN model is represented as $(N_{row} \times N_{col}) = (10 \times 10)$ grids over 201 frequency points linearly distributed between 200MHz and 20GHz, which gives $\mathbf{100 \times 100 \times 201 \approx 2}\boldsymbol{M}$ impedances to be evaluated per each task; the PDN scale is reasonable to reflect the simulation intensiveness of DPP. The RLGC electrical parameters of PDN unit-cell are shown in Appendix A.2. Out of the $N_{row} \times N_{col}$ ports on PDN, one is assigned as a probing port and 0 to 15 ports are assigned as keep-out ports (see Appendix A.4). Decap is modeled as a unit-cell with a single port that is attached to a specific port on PDN when an action is made. The RLGC electrical parameters of decap unit-cell are also shown in Appendix A.2. 100 PDN cases for test and another 100 PDN cases for validation were generated for performance evaluation. We made sure test data, validation data and training data did not overlap. We used the number of decap $K = 20$ for every training, but K can be changed during the inference.

Offline Expert Data Collection. Since expert data for the DPP benchmark was not available, we synthetically generated offline expert data using genetic algorithm (GA) for this study. The number of iterations done for collecting a single data is represented as M. Note that the higher the M, the better the quality of data, but the higher the simulation cost. We used GA$\{M = 100\}$ to collect the offline expert data. In addition, we denote N as the number of offline expert data used for training CSE. Note that the lower the N the more data-efficient the training is.

Hyperparameters. For training, we generate three transformed data per expert data. We denote $P(= 3)$ as the number of AP-transformed data per offline data. Thus, the total number of guiding data becomes $N \times (P + 1)$. For instance, $P = 3, N = 50$ makes total $50 \times 3 + 50 = 200$ guiding data. We set the distribution ρ, described in Section 2.1, as uniform distribution for training.

We used $N = 2000$ offline expert data for training CSE and imitation learning-based baselines. For the learning algorithm, we used ADAM (Kingma & Ba, 2015) with a learning rate of 10^{-5}. We trained our model with batch size 100 for $N < 200$ and batch size $1,000$ for $N = 1,000$ and $2,000$. We trained for a maximum of 200 epochs for each model; we used the model with the best validation score for CSE and other ML baselines to evaluate performance. See Appendix D for detailed setup.

Baselines for Comparison. For search heuristic baseline methods, we implemented random search and genetic algorithm. Since these methods are iterative solvers, they require a large number of simulations (i.e., $M >= 100$. See Table 1) for each problem. For non-iterative learning-based solvers, we reported two RL baselines, AM-RL (Park et al., 2022) and Arb-RL (Kim et al., 2021) and two IL baselines, AM-IL and Arb-IL, which are modified AM-RL and Arb-RL with imitation learning instead of reinforcement learning, to investigate the effectiveness of CSE components compared to the same imitation learning approaches with different learning strategies/neural architecture. Implementation details of the baselines are provided in Appendix D.2 and Appendix D.3.

5.2 Generalization Capability Evaluation

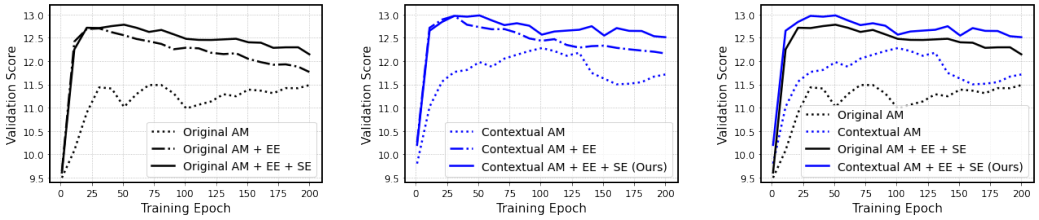
To verify the generalization capability of the trained solver, each method is given the same unseen 100 DPPs and the average performance score was measured, after allocating $K = 20$ decaps on each.

As shown in Table 1, our CSE significantly outperformed all baselines in terms of average performance score. 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 learning-based baselines and CSE do not require simulations to generate solutions; once trained, they only require a single simulation to measure the performance. Though learning-based methods can easily find such a solution, CSE is the only method capable of finding a solution that outperforms the highly iterative online search methods by a zero-shot inference.

When the number of costly simulations was limited, RL-based methods (AM-RL, Arb-RL) showed poorer generalization capability than their imitation learning versions (AM-IL, Arb-IL) due to inefficiency in exploring over extremely large combinatorial action space of DPP. We believe that 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 D.2. Note that if we have an infinite budget for reward simulation (which never happens in a real-world hardware setting), DRL could achieve greater performance and generalization capability with a sufficient learning loop.

Among the imitation learning approaches trained with the same number of offline expert data ($N = 2,000$), CSE showed the highest performance. We believe that such higher generalization capability 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) the neural architecture with PCN and RCN makes the policy easily adapt to new task conditions.

Extrapolation over Expert Method. The CSE policy trained with offline expert data generated by the expert policy, GA$\{100\}$, outperformed GA$\{500\}$, with zero-shot inference. That is, the CSE policy trained with low-quality offline expert data produced higher-quality designs. We believe this was possible because we trained a factorized form of 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. In addition, the CSE with symmetric label transformation has further guided the policy to learn such an effective decap placement design scheme.



(a) CSE in Original AM (b) CSE in Contextual AM (c) Original vs. Contextual AM

Figure 5: Ablation study on CSE components

Ablation Study. We conducted ablation studies to validate the effectiveness of CSE components and context neural networks with sparse offline data ($N = 100$). We ablated the effectiveness of expert exploitation (EE) and self-exploitation (SE) in two policy networks: original AM and contextual AM (ours). The original AM refers to the AM-IL baseline. Each component of CSE supported increasing generalization capability in both policy networks and the contextual AM with newly devised context neural networks was verified to outperform the original AM. Therefore, we verified that both CSE components and the modifications of AM successfully contributed to the promising performance.

5.3 Offline Data Efficiency Evaluation

N is the number of offline expert data generated by the expert method, GA $\{M = 100\}$. We ablated $N \in \{100, 500, 1000, 2000\}$ with fixed $P = 3$ and $\lambda = 8$ and compared to the AM-IL baseline. As shown in Fig. 6, outperformed AM-IL baseline in all N variation and CSE with $N = 100$ achieved a higher validation score than AM-IL with $N = 2000$. In addition, the performance of AM-IL saturates at $N > 500$ while the performance of continuously increases with the increase of N.

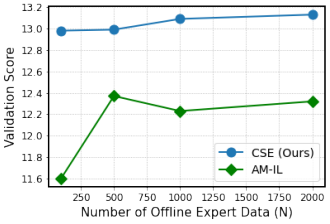


Figure 6: Offline data-efficiency evaluation

Table 2: Scalability evaluations on larger PDN scale and varying number of decap K.

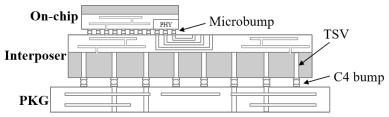
Scale Variables		Methods		
PDN Scale	Number of Decap, K	GA $\{100\}$	AM (Park et al., 2022)+IL	**CSE (ours)**
10×10	12	11.77	10.22	**12.23**
	16	12.25	11.13	**12.60**
	20	12.53	11.71	**12.81**
	24	12.79	12.20	**12.95**
	30	13.02	12.62	**13.11**
15×15	20	7.61	6.23	**8.47**
	40	7.69	7.75	**8.54**

5.4 Scalability Evaluation

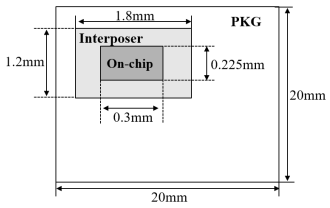
For scalability verification, learning-based DPP methods were pre-trained for a fixed scale PDN, (10×10), and a fixed number of decaps, $K = 20$. Then, the pre-trained models were asked to place decaps of varying $K \in \{12, 16, 20, 24, 30\}$ on (10×10) PDN and varying $K \in \{20, 40\}$ for a larger (15×15) PDN without additional training (i.e, zero-shot). We chose two baseline methods for comparison: GA $\{100\}$ and AM-IL. As shown in Table 2, our CSE outperformed GA $\{100\}$ and AM-IL for all scales. Furthermore, CSE 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 cost.

5.5 Flexibility Verification for Practical Application

To verify the practical applicability, we applied the proposed CSE to a real-world hardware application, high bandwidth memory (HBM), which an interposer-based 2.5D. As shown in Fig. 7, the hierarchical PDN model of HBM is composed of (40×40) package PDN, (40×60) interposer PDN and (15×20) on-chip PDN, each layers connected by TSV + C4 bumps and microbumps.



(a) Sideview



(b) Topview

Figure 7: Structure of HBM PDN model

For performance evaluation, we compared the CSE trained with $N = 1000$ offline expert data to GA$\{M = 100\}$ on placing varying number of decaps, $K \in \{12, 14, 16, 18, 20, 22, 24, 26, 28, 40\}$ for 100 test cases. The pre-trained solver with $K = 20$ was used for CSE without additional training.

Fig. 8 compares the performances of CSE and GA$\{M = 100\}$ and verifies that CSE achieves higher performance with significantly fewer decaps. For instance, the performance score of 26.68, attained with 40 decaps by GA$\{M = 100\}$, was achieved with only 26 decaps placed by zero-shot CSE. Power noise analysis in a test case was carried out (see Appendix E.3), where the initial power noise before placing decap was 10.546mV. CSE was able to reduce the power noise to 0.610mV (94.2% reduction) with 26 decaps while GA$\{M = 100\}$ reduced to 0.682mV (93.5% reduction) with 40 decaps. Reducing the number of decaps is a huge contribution to the industry as hardware devices are mass-produced, reducing a single decap can greatly reduce production costs.

6 Conclusion

This paper proposed the *collaborative symmetricity exploitation* (CSE) framework for training contextualized policy (i.e., solver) of placement tasks in an offline manner. The CSE was applied to decap placement problem (DPP) and achieved the most promising performance among all baseline methods. The CSE was also validated on several scales and on multiple hardware devices. The CSE is a general purpose offline learning scheme for placement tasks that can be further applied to other hardware placement tasks including chip placement, ball grid array (BGA) placement, and via placement.

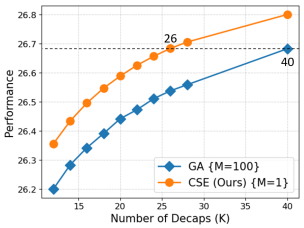


Figure 8: Performance comparison with number of decap variation on HBM

REFERENCES

Anthony Agnesina, Kyungwook Chang, and Sung Kyu Lim. 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.

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

Emmanuel Bengio, Moksh Jain, Maksym Korablyov, Doina Precup, and Yoshua Bengio. Flow network based generative models for non-iterative diverse candidate generation. *Advances in Neural Information Processing Systems*, 34:27381–27394, 2021.

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

Kyungjun Cho, Youngwoo Kim, Subin Kim, Hyunwook Park, Junyong Park, Seongsoo Lee, Daeyong Shim, Kangseol Lee, Sangmook Oh, and Joungho Kim. Fast and accurate power distribution network modeling of a silicon interposer for 2.5-d/3-d ics with multiarray tsvs. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 9(9):1835–1846, 2019. doi: 10.1109/TCPMT.2019.2895083.

Francesco de Paulis, Riccardo Cecchetti, Carlo Olivieri, and Markus Buecker. 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.

Ihsan Erdin and Ramachandra Achar. 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.

Jun Fan, J.L. Knighten, A. Orlandi, N.W. Smith, and J.L. Drewniak. 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.

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

Winston Haaswijk, Edo Collins, Benoit Seguin, Mathias Soeken, Frédéric Kaplan, Sabine Süsstrunk, and Giovanni De Micheli. 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.

Abdelrahman Hosny, Soheil Hashemi, Mohamed Shalan, and Sherief Reda. 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.

Jisoo Hwang, Jun So Pak, Dooseok Yoon, Heeseok Lee, James Jeong, Yun Heo, and Ilryong Kim. 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.

Moksh Jain, Emmanuel Bengio, Alex Hernandez-Garcia, Jarrid Rector-Brooks, Bonaventure FP Dossou, Chanakya Ajit Ekbote, Jie Fu, Tianyu Zhang, Michael Kilgour, Dinghuai Zhang, et al. Biological sequence design with gflownets. In *International Conference on Machine Learning*, pp. 9786–9801. PMLR, 2022.

Jack Juang, Ling Zhang, Zurab Kiguradze, Bo Pu, Shuai Jin, and Chulsoon Hwang. 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.

Haeyeon Kim, Hyunwook Park, Minsu Kim, Seonguk Choi, Jihun Kim, Joonsang Park, Seongguk Kim, Subin Kim, and Joungho Kim. 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.

Jaemin Kim 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.

Minsu Kim, Junyoung Park, and Jinkyoo Park. Sym-nco: Leveraging symmetricity for neural combinatorial optimization. *arXiv preprint arXiv:2205.13209*, 2022.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL `http://arxiv.org/abs/1412.6980`.

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

Yeong-Dae Kwon, Jinho Choo, Byoungjip Kim, Iljoo Yoon, Youngjune Gwon, and Seungjai Min. Pomo: Policy optimization with multiple optima for reinforcement learning. *Advances in Neural Information Processing Systems*, 33, 2020.

Haiguang Liao, Wentai Zhang, Xuliang Dong, Barnabas Poczos, Kenji Shimada, and Levent Burak Kara. 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.

Haiguang Liao, Qingyi Dong, Weiyi Qi, Elias Fallon, and Levent Burak Kara. 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.

Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, Jiwoo Pak, Andy Tong, Kavya Srinivasa, William Hang, Emre Tuncer, Quoc Le, James Laudon, Richard Ho, Roger Carpenter, and Jeff Dean. A graph placement methodology for fast chip design. *Nature*, 594:207–212, 06 2021. doi: 10.1038/s41586-021-03544-w.

Hyunwook Park, Junyong Park, Subin Kim, Daehwan Lho, Shinyoung Park, Gapyeol Park, Kyungjun Cho, and Joungho Kim. Reinforcement learning-based optimal on-board decoupling capacitor design method. In *2018 IEEE 27th Conference on Electrical Performance of Electronic Packaging and Systems (EPEPS)*, pp. 213–215, 2018. doi: 10.1109/EPEPS.2018.8534195.

Hyunwook Park, Junyong Park, Subin Kim, Kyungjun Cho, Daehwan Lho, Seungtaek Jeong, Shinyoung Park, Gapyeol Park, Boogyo Sim, Seongguk Kim, Youngwoo Kim, and Joungho Kim. 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, 2020. doi: 10.1109/TCPMT.2020.2972019.

Hyunwook Park, Minsu Kim, Seongguk Kim, Keunwoo Kim, Haeyeon Kim, Taein Shin, Keeyoung Son, Boogyo Sim, Subin Kim, Seungtaek Jeong, et al. Transformer network-based reinforcement learning method for power distribution network (pdn) optimization of high bandwidth memory (hbm). *arXiv preprint arXiv:2203.15722*, 2022.

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

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. V.

Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (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.

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

Zhifei Xu, Zihao Wang, Yin Sun, Chulsoon Hwang, Hervé Delingette, and Jun Fan. 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.

Ling Zhang, Wenchang Huang, Jack Juang, Hank Lin, Bin-Chyi Tseng, and Chulsoon Hwang. 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.

Zhenxin Zhao and Lihong Zhang. 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.

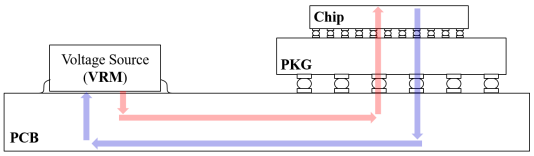
A DPP ELECTRICAL MODELING AND PROBLEM DEFINITION

This section provides electrical modeling details of PDN and decap models used for verification of CSE in DPP. Note that these electrical models can be substituted by those of your 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 3. 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 Intel i7. Note that simulation time depends on the size and complexity of the PDN model.

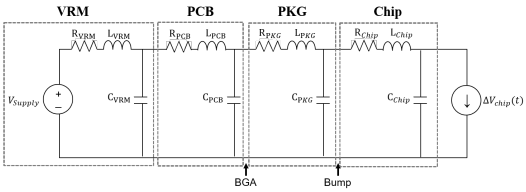
Table 3: Time Taken for an Objective Evaluation of a PDN model described in Appendix A.2

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

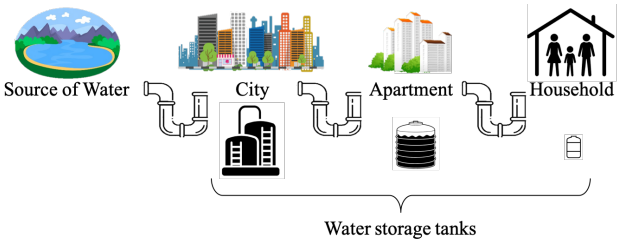
A.1 DOMAIN PERSPECTIVE DECAP PLACEMENT PROBLEM



(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 9: Illustration of Hierarchical Power Distribution Network (PDN) analogous to Water Supply Chain.

The development of AI has led to an increased demand for high-performance computing systems. High-performance computing systems not only require 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 CMOS transistors (Hwang et al., 2021).

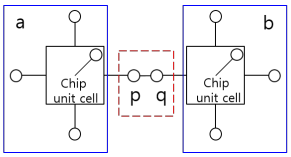
Fig. 9 (a) shows the power distribution network (PDN) consisting of all the power/ground planes from the voltage source to operating chips. Power is generated in VRM and delivered through

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 the reliable power supply to the operating chips, thus improving the power integrity of hardware. As shown in Fig. 9 (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 if placing more water tanks can make the water supply more stable, placing more decaps can make power supply more reliable. However, because adding more decaps requires more space and is costly, optimally placement of decaps is important in terms of PI and cost/space-saving.

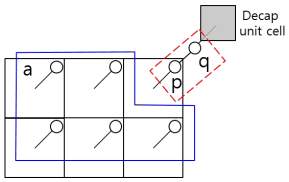
A.2 PDN AND DECAP MODELS FOR VERIFICATION

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 equation-based python implemented segmentation method, illustrated in Fig. 10.

Segmentation method was used for generation of PDN model consisting of a chip layer and a package layer for verification as illustrated in Fig. 10 (a). The segmentation method was also used for 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. 10 (b).



(a) Generation of Chip PDN.



(b) Decap Placement.

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

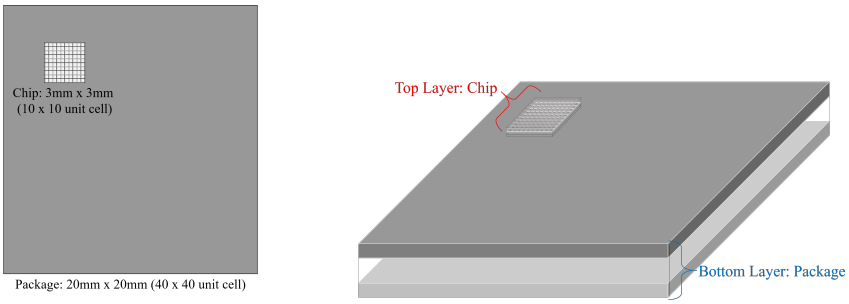
$$\begin{bmatrix} V_a \\ V_p \end{bmatrix} = \begin{bmatrix} Z_{aa} & Z_{ap} \\ Z_{pa} & Z_{pp} \end{bmatrix} \begin{bmatrix} I_a \\ I_p \end{bmatrix} \qquad \begin{bmatrix} V_b \\ V_q \end{bmatrix} = \begin{bmatrix} Z_{bb} & Z_{bq} \\ Z_{qb} & Z_{qq} \end{bmatrix} \begin{bmatrix} I_b \\ I_q \end{bmatrix}$$

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

(c) Segmentation Method.

Figure 10: Segmentation Method Implemented for PDN Generation and Decap Placement on PDN.

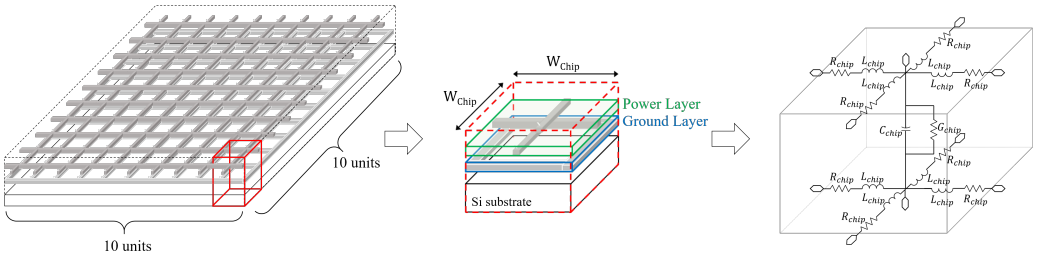
The PDN model we used for verification has a two-layer structure; a package layer at the bottom and a chip layer on top of it as illustrated in Fig. 11. The PDN was modeled through the unit-cell segmentation method. Package layer was composed of 40×40 package unit-cells and chip layer was composed of 10×10 (i.e, $N_{row} \times N_{col}$) chip unit-cells. Because the DPP benchmark places MOS type decaps, which are placed on chip, ports are only available on chip. Thus, we extracted 10×10 ports information from the chip layer. See Fig. 14 (a), illustrating the chip PDN divided into 10×10 units and each unit-cell numbered.



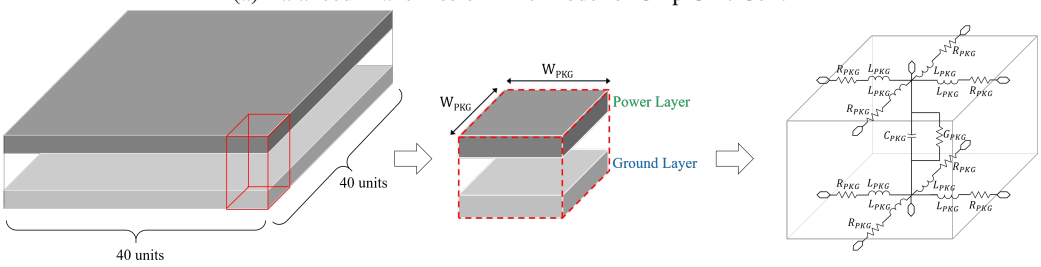
(a) Top-View of PDN model. (b) Side-View of PDN model.

Figure 11: Top-view and Side-view of PDN Model used for Verification

The electrical models of package and chip unit-cells that are used to build the PDN model for verification are described in Fig. 12. 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 4.



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



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

Figure 12: Electrical Modeling of Chip and Package Unit-Cells for PDN Model generation.

Table 4: 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. Decap model and its electrical parameters are shown in Fig. 13. As mentioned in appendix A.1 Fig. 10 (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 CSE, these PDN and decap models can be substituted by those of your interests.

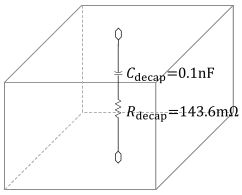
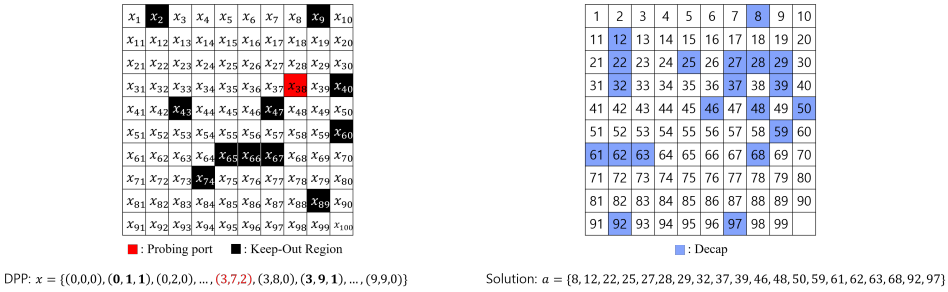


Figure 13: Decap Unit-Cell with the Electrical Parameters used for Verification.

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



(a) Input Problem PDN.

(b) Output Decap Placement Solution.

Figure 14: Illustration of how the DPP problem with specific condition is given as an input and decap placement solution is generated as an output.

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 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. 14 (b), the solution is given as a set of port numbers corresponding to each decap location.

A.4 Random Problem Generation of DPP

To randomly generate decap placement problems (DPPs) with distinct conditions for training, test 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

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, the objective of which is to find the placement of 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 test dataset was generated. We made sure the guiding data problems do not overlap with the test dataset problems. Also, we made sure each guiding problem does not overlap with each other. Each guiding data problem goes through the following process described in Fig. 15 to collect the corresponding expert label.

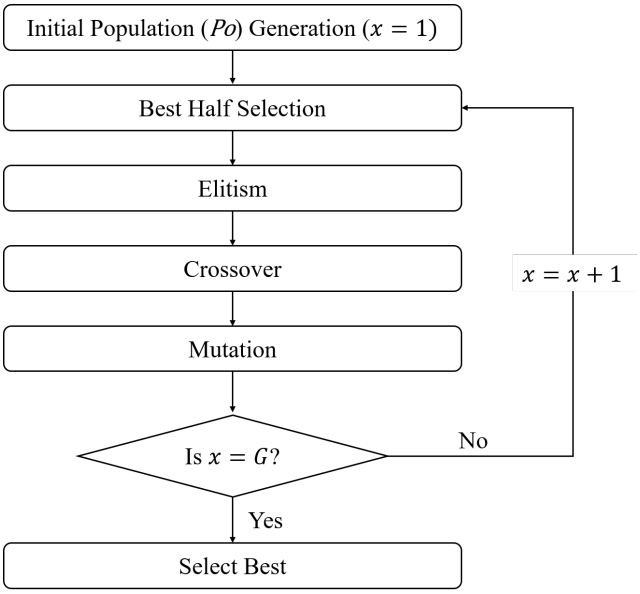


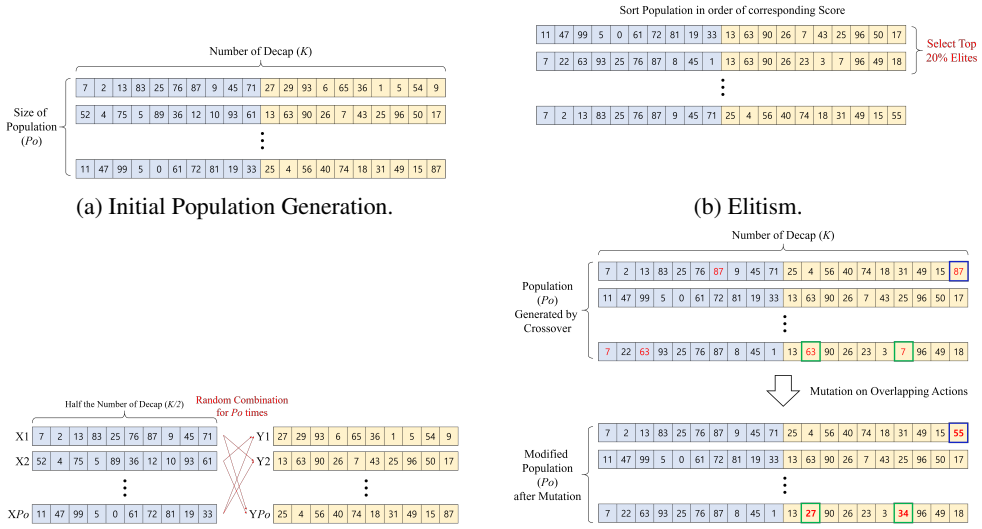
Figure 15: Process Flow of Genetic Algorithm for DPP.

Population and Generation. For GA $\{M = 100\}$ (*expert policy*), we fixed the size of population as $P_0 = 20$ and the number of generation as $G = 5$, which makes up 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. 14 (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 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. 16 (c), half the solutions on the left and the other half on the right go through random crossover for P_0 times to



(a) Initial Population Generation.

(b) Elitism.

(c) Crossover.

(d) Mutation.

Figure 16: Illustration of each GA Operators used for DPP Guiding Data Generation.

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. 16 (d), there may exist solutions with overlapping numbers 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 as a result of GA are saved and used as guiding expert labels for imitation learning.

C DETAILS OF NEURAL ARCHITECTURE DESIGN

Our neural architecture has the AM (Kool et al., 2019) with context modification. The AM is a transformer(Vaswani et al., 2017)-based encoder-decoder model designed to solve combinatorial optimization problems. We used conventional notations from transformer (Vaswani et al., 2017) and AM (Kool et al., 2019), including multi-head attention (MHA), feed forward (FF), query, key and value (Q, K, V). Because their terminologies are well organized, we tried to keep every notation as possible. In this paper, we focused on presenting the main differences between AM and our architecture. See Kool et al. (2019) for detailed mechanism of AM.

C.1 CHANGE OF NOTATIONS.

There are small revisions we made from Kool et al. (2019). In AM, TSP nodes are presented as $\boldsymbol{x}_i$, $i \in \{1, ..., N\}$, where N refers to the number of TSP nodes. This paper uses I_{probe} for the node of the probing port, $I_{keepout}$ for nodes of the keep-out regions and $I_{allowed}$ for nodes of the decap-allowed ports.

Kool et al. (2019) denotes action as $\boldsymbol{\pi}$ (for representing permutation action), but we denoted action as $\boldsymbol{a}$.

In, Kool et al. (2019), the notation, $\boldsymbol{h}^{(N)}$, refers to N times MHA in encoder; we denoted this notation as $\boldsymbol{h}$ just for readability.

There are two additional notations: $\boldsymbol{c}_{probe}$ is the probing context embedding from the probing port context network (PCN in section 3.2) and $\boldsymbol{c}_{a_{t-1}}$ is the recurrent context embedding from the recurrent context network (RCN in section 3.2) for $step = t$.

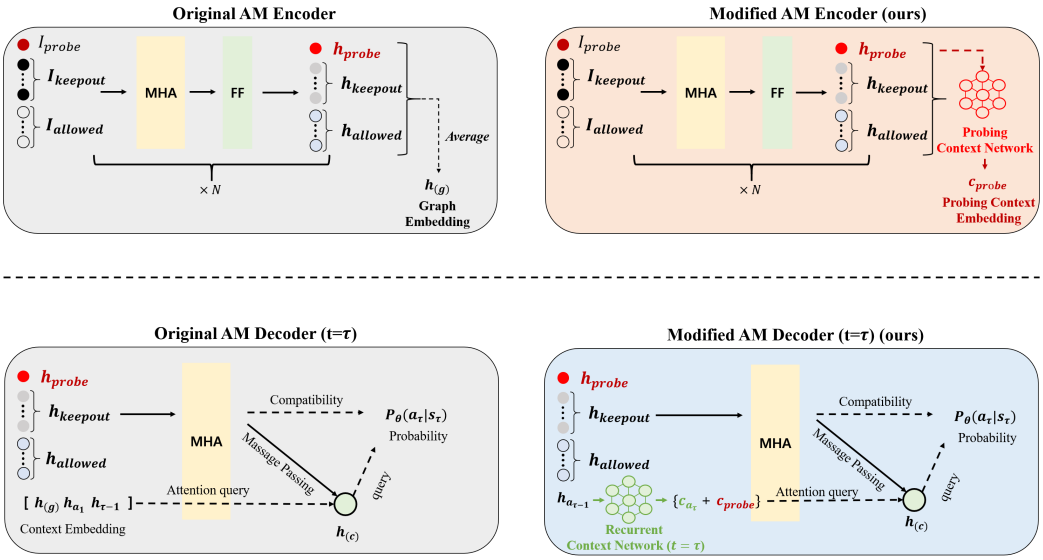


Figure 17: Overview of main difference between AM and modified version of AM.

C.2 HIGHLIGHT OF MODIFICATIONS: CONTEXT EMBEDDING.

The main difference between the AM and ours is the context embedding and is illustrated in Fig. 17.

AM's (Kool et al., 2019) context embedding is presented as follows:

$$\boldsymbol{h}_{(\boldsymbol{c})} = MHA([\boldsymbol{h}_{(\boldsymbol{g})}, h_{a_{\tau-1}}, \boldsymbol{h}_{\boldsymbol{a_1}}], \boldsymbol{h}) \tag{7}$$

Context embedding of AM. Since the AM was originally designed for TSP and its invariant problems, AM's context embedding is implemented for capturing the entire graph by taking the average of all

node embedding, $\boldsymbol{h_{(g)}}$, state-transition with $\boldsymbol{h_{a_{\tau-1}}}$ and final destination with $\boldsymbol{h_{a_1}}$. Note that TSP is a routing problem, where it must return to the first node (i.e, destination node is first visited node).

Context embedding of AM for DPP (AM-RL (Park et al., 2022)). Park et al. (2022) also used the AM for decap placement with modification of context embedding. Park et al. (2022) tried to add $\boldsymbol{h_{probe}}$ to capture the location of probing port as follows:

$$\boldsymbol{h_{(c)}} = MHA([\boldsymbol{h_{(g)}}, \boldsymbol{h_{a_{t-1}}}, \boldsymbol{h_p}], \boldsymbol{h}) \tag{8}$$

Context embedding of Ours. We observed that $\boldsymbol{h_{(g)}}$ degrades the performance of the model for DPP. DPP is different from TSP; we need a new DPP-specific context embedding strategy. Therefore, we tried to focus on the probing port more than others by proposing the PCN. We removed $\boldsymbol{h_{(g)}}$ and $\boldsymbol{h_{a_1}}$ from the context embedding and replaced them with our newly designed context embedding. Our context embedding is described as follows:

$$\boldsymbol{h_{(c)}} = MHA(\boldsymbol{c_{probe}} + \boldsymbol{c_{a_{t-1}}}, \boldsymbol{h}) \tag{9}$$

$$\boldsymbol{c_{probe}} = \mathbf{MLP}_{PCN}(\boldsymbol{h_{probe}}) \tag{10}$$

$$\boldsymbol{c_{a_{t-1}}} = \mathbf{MLP}_{RCN}(\boldsymbol{h_{a_{t-1}}}) \tag{11}$$

Note that both $\mathbf{MLP}_{PCN}$ and $\mathbf{MLP}_{RCN}$ are two-layer perceptron models with ReLU activation, where input and output dimensions are identical ($d = 128$ in all experiments).

C.3 Calculation of Probability.

Probability calculations using context hidden embedding $\boldsymbol{h}_{(c)}$, and PDN hidden embedding $\boldsymbol{h}_i$, $i \in \{1, ..., N_{row} \times N_{col}\}$ in (11-14) are exactly identical to (5-8) in Kool et al. (2019) except the masking mechanism in equation 13 and equation 14. Because Kool et al. (2019) solves TSP, so they mask the previously selected actions by forcing $-\infty$ as compatibility $u_{(c)j}$. For DPP, we mask not only the previously selected actions $\boldsymbol{a}_{1:t-1}$ but also the probing port index I_{probe} and the keep-out region indices $I_{keepout}$; it is forbidden to choose the I_{probe}, $I_{keepout}$ and previously selected actions $a_{1:t-1}$

Query, key and value are computed by:

$$\boldsymbol{q}_c = W^Q \boldsymbol{h}_{(c)}, \boldsymbol{k}_i = W^K \boldsymbol{h}_i, \boldsymbol{v}_i = W^V \boldsymbol{h}_i \tag{12}$$

Note that W^Q, W^K and W^V are 128-to-128 linear projections.

After that, compatibility $u_{(c)j}$ is computed by the dot product of query and key, with masking mechanism (setting $-\infty$ not to select actions in $\boldsymbol{s}_{t-1}$).

$$u_{(c)j} = \begin{cases} \frac{\mathbf{q}_{(c)}^T \mathbf{k}_j}{\sqrt{128}} & \text{if } j \notin I_{probe}, I_{keepout}, a_{1:t-1} \\ -\infty & \text{otherwise} \end{cases} \tag{13}$$

The $tanh$ clipping is done following Bello et al. (2016) and Kool et al. (2019).

$$u_{(c)j} = \begin{cases} 10 \cdot \tanh\left(\frac{\mathbf{q}_{(c)}^T \mathbf{k}_j}{\sqrt{128}}\right) & \text{if } j \notin I_{probe}, I_{keepout}, a_{1:t-1} \\ -\infty & \text{otherwise.} \end{cases} \tag{14}$$

Finally, probability can be computed using softmax function as follows:

$$p_{\boldsymbol{\theta}}\left(a_t = i \mid \boldsymbol{s}_t\right) = \frac{e^{u_{(c)i}}}{\sum_j e^{u_{(c)j}}} \tag{15}$$

D DETAILED EXPERIMENTAL SETTINGS

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

D.1 TRAINING HYPERPARAMETERS.

There are several hyperparameters for training; we tried to fix the hyperparameters as Kool et al. (2019) did for showing 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 5: Hyperparameter setting for training model.

Hyperparameter	Value
learning rate	0.00001
λ	8×10^{32}
N	1000
P	3
B	1000

D.2 IMPLEMENTATION OF ML BASELINES.

There are two main ML baselines, Arb-RL (Kim et al., 2021) and AM-RL (Park et al., 2022).

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

Arb-IL. Arb-IL is an imitation learning version of Arb-RL trained by our training data. We set $N = 2000$, $B = 1000$ for training Arb-IL.

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

AM-IL. AM-IL is an imitation learning version of AM-RL trained by our training data. For experiments in Table 1, we set $N = 2000$ and $B = 1000$ for training. For 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 baselines and CSE:

Table 6: Training sample complexity of ML baselines and CSE.

Methods	The Number of PI simulations for Training
Arb-RL	200,000
AM-RL	200,000
Arb-IL $\{N = 2000\}$	200,000 ($N = 2000$, $M = 100$ from GA expert)
AM-IL $\{N = 2000\}$	200,000 ($N = 2000$, $M = 100$ from GA expert)
CSE $\{N = 100\}$ (ours)	10,000 ($N = 100$, $M = 100$ from GA expert)
CSE $\{N = 1000\}$ (ours)	100,000 ($N = 1000$, $M = 100$ from GA expert)
CSE $\{N = 2000\}$ (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).

[1] https://github.com/pemami4911/neural-combinatorial-rl-pytorch

[2] https://github.com/wouterkool/attention-learn-to-route

D.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 CSE. 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.

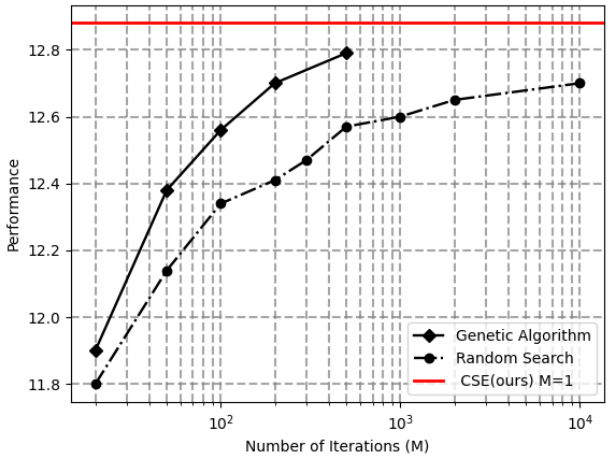


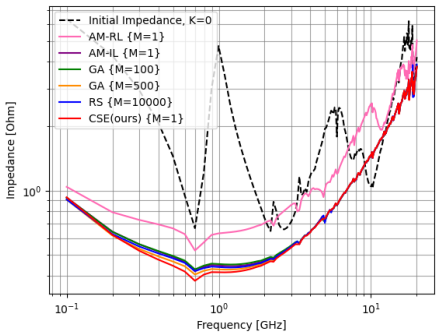
Figure 18: Performance of GA and RS with varying number of iterations (M) in comparison to CSE at $M = 1$.

Fig. 18 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 CSE showed higher performance than GA$\{M = 100\}$ and RS $\{M = 10,000\}$ with a single inference $M = 1$.

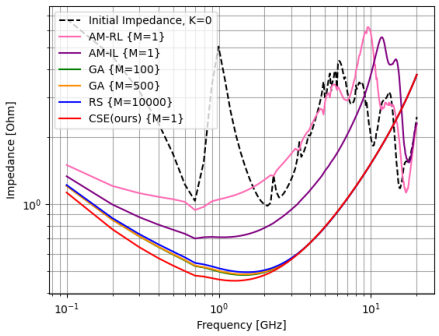
E EXPERIMENTAL RESULTS IN TERMS OF POWER INTEGRITY

The objective of DPP is to suppress impedance of the probing port 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 CSE was evaluated in comparison to GA $\{M = 100\}$ (*expert policy*), GA $\{M = 500\}$, RS $\{M = 10,000\}$, AM-RL and AM-IL on unseen 100 PDN cases. Each method was asked to place 20 decaps ($K = 20$) on each test.

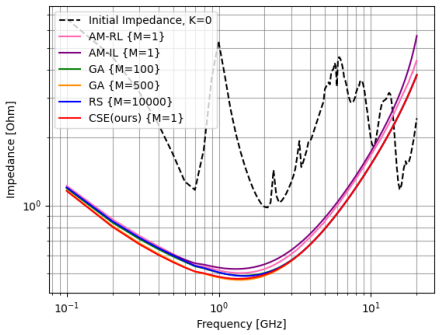
E.1 IMPEDANCE SUPPRESSION PLOTS



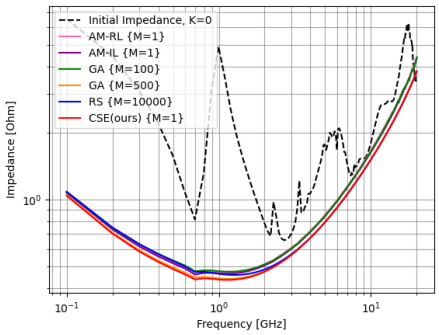
(a) Test Case 1.



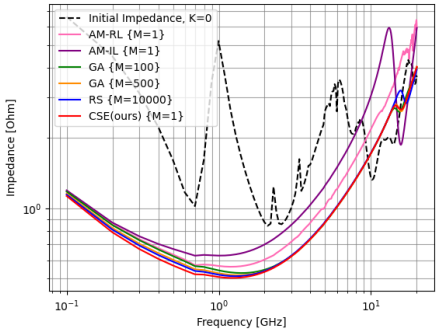
(b) Test Case 2.



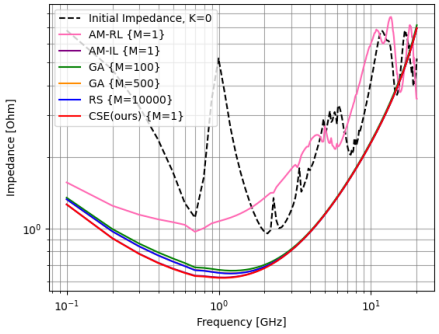
(c) Test Case 3.



(d) Test Case 4.



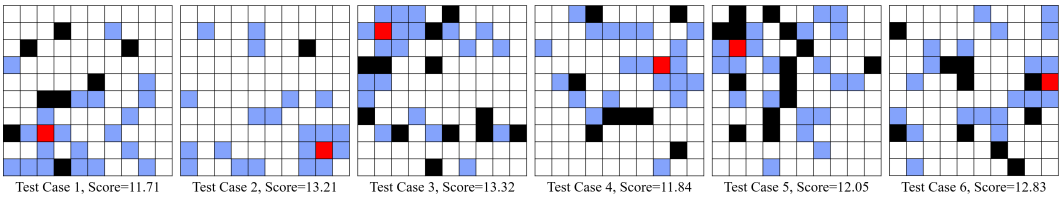
(e) Test Case 5.



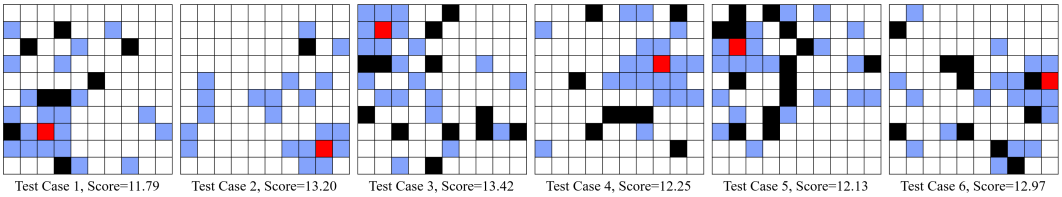
(f) Test Case 6.

Figure 19: Impedance suppressed by each method, GA $\{M = 100\}$ (*expert policy*), GA $\{M = 500\}$, RS $\{M = 10,000\}$, AM-RL , AM-IL and CSE (Ours) for 6 example PDN cases out of 100 test dataset. (The lower the better.)

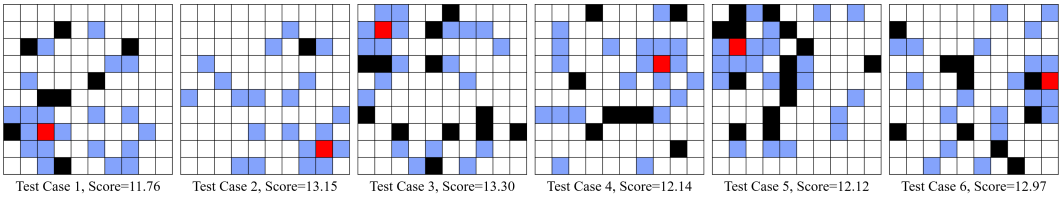
E.2 DECAP PLACEMENT TENDENCY ANALYSIS



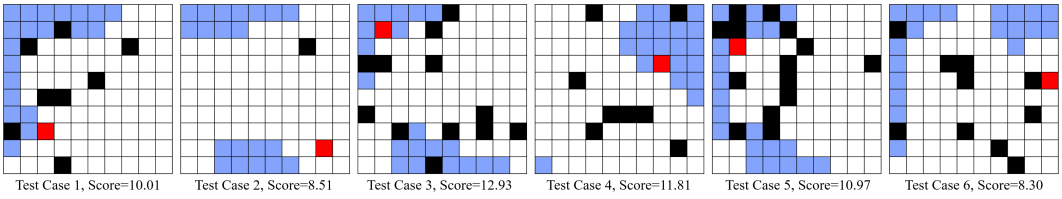
(a) GA $\{M = 100\}$.



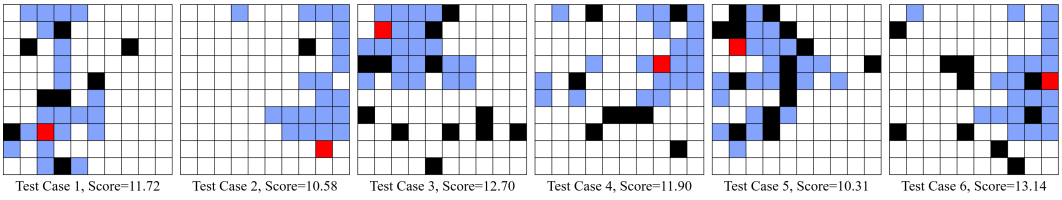
(b) GA $\{M = 500\}$.



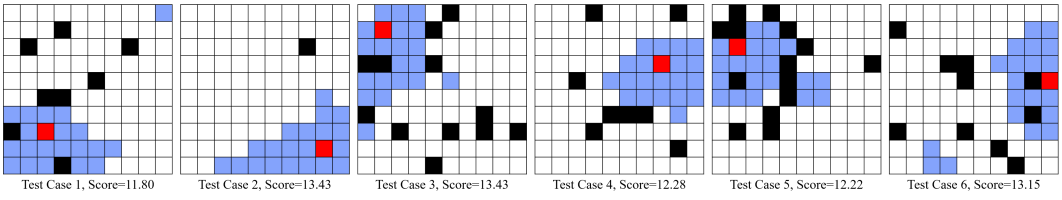
(c) RS $\{M = 10,000\}$.



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



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



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

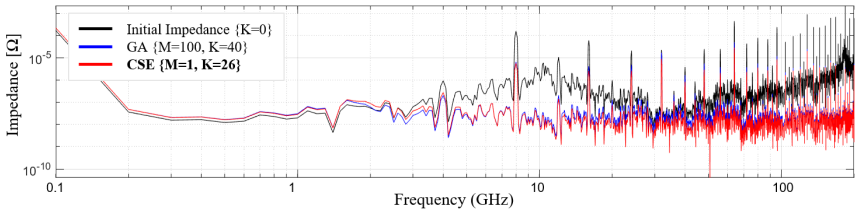
Figure 20: Corresponding decap placement solutions to Fig. 19 by each method. Red represents probing port, black represents keep-out ports and blue represents decap locations.

Fig. 21 shows the decap placement solutions of 6 PDN cases plotted in Fig. 19. The solutions by the search-heuristic methods, GA and RS, tend to be scattered while the solutions by learning-based methods, AM-RL, AM-IL and CSE, are clustered. Since search-heuristic methods are based on

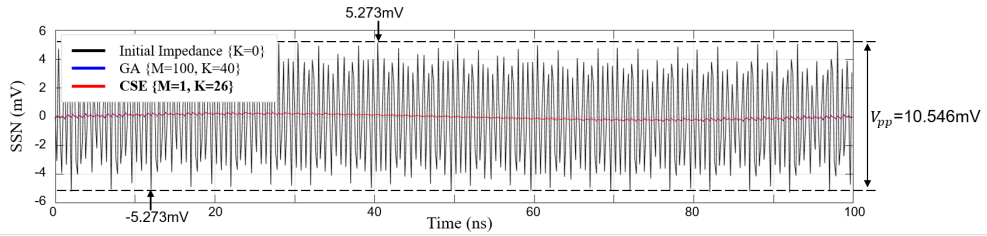
random generations, they do not show clear tendency. On the other hand, learning based methods are based on a policy so that they have distinct tendency in placing decaps.

The role of placing decaps in hardware design is to decouple 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 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, CSE understands the PDN structure and its electrical properties by data-driven learning. According to Fig. 21, CSE tends to place decaps near the probing port, which is a well-known expert rule in the PI domain.

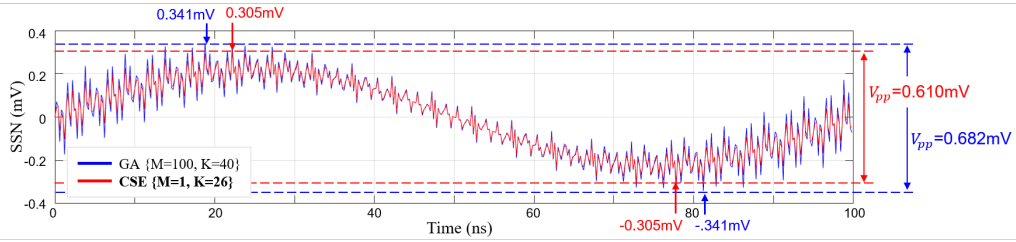
E.3 Power Noise Analysis on HBM PDN



(a) Impedance suppression.



(b) Initial power noise before decap placement.



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

Figure 21: Power noise analysis in terms of simultaneous switching noise (SSN) on HBM PDN before and after decap placed by our CSE$\{M = 1, K = 26\}$ and GA$\{M = 100, K = 40\}$.

Appendix E.3 analyzes the performance of CSE 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 CSE placed 26 decaps and GA$\{M = 100\}$ placed 40 decaps. CSE reduced power noise more than GA$\{M = 100\}$ with 14 less decaps. The impedances of the probing port on power distribution network (PDN) before and after decap placed by CSE and GA$\{M = 100\}$ are presented in Fig. 21a. The time-domain power noise before and after decap placement by each method is shown in Fig. 21b. For performance comparison, Fig. 21c shows the time-domain power noise after decap placement by CSE and GA$\{M = 100\}$.

F FURTHER ABLATION STUDY

This section reports ablation studies on action permutation invariance and hyperparameters N (number of guiding samples), λ (weight of self-exploitation loss term), and P (number of permutation transformed labels).

F.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 $P = 3$ and $\lambda = 8$ and compare to AM-IL baseline for all N. As shown in Table 7, CSE with $N = 2000$ gives the best performance and CSE outperforms AM-IL for all N variations. Performance of AM-IL is saturated at $N > 500$ while the performance of CSE continuously increases with the increase of N.

Table 7: Ablation study on N for CSE ($P = 3, \lambda = 8$) and AM-IL.

	Validation Score
AM-IL $\{N = 100\}$	11.60
CSE (ours) $\{N = 100\}$	**12.98**
AM-IL $\{N = 500\}$	12.37
CSE (ours) $\{N = 500\}$	**12.99**
AM-IL $\{N = 1000\}$	12.23
CSE (ours) $\{N = 1000\}$	**13.09**
AM-IL $\{N = 2000\}$	12.32
CSE (ours) $\{N = 2000\}$	**13.13**

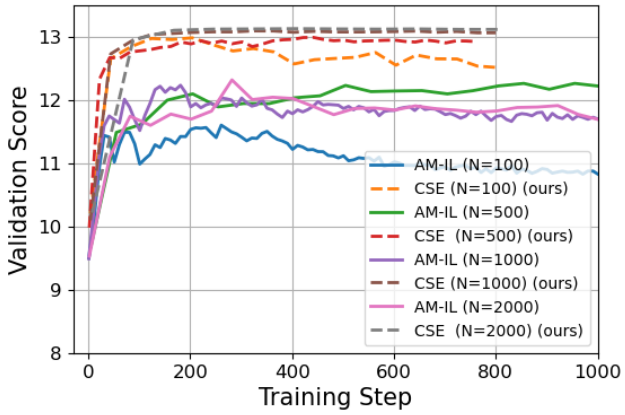


Figure 22: Validation graph CSE in comparison to AM-IL for varying number of offline expert data $N \in \{100, 500, 1000, 2000\}$.

F.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 $\lambda \times L_U$ 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 $N = 100$ and $P = 3$. For every λ, it prevents overfitting of the model in comparison to the baselines trained only with L_{Expert} (see Fig. 23). According to the Table 9, $\lambda = 8$ gives the best validation scores.

Table 8: Ablation study of λ on fixed $P = 3$ and $N = 100$.

λ ($\times 10^{32}$)	Validation Score
1	12.96
2	12.96
4	12.94
6	12.96
7	12.98
8	**12.98**
9	12.97
10	12.96
Only IL, $\lambda = 0$	12.97

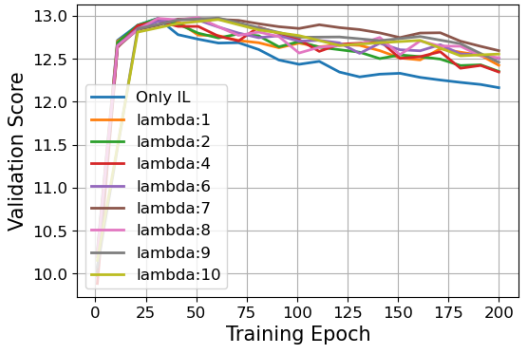


Figure 23: Validation graph of $\lambda \in \{1, 2, 4, 6, 7, 8, 9, 10\}$ on fixed $P = 3$ and $N = 100$.

F.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 $P \in \{3, 5, 7\}$ with fixed $N = 100$ and $\lambda = 8$ and compared collaborative symmetricity exploitation (i.e., both expert and self-exploitation) to only expert exploitation training case. As shown in Table 9, $P = 3$ with {Expert exploitation + Self-exploitation} give best performances. For every P, {Expert exploitation + Self-exploitation} gives the better performances, indicating self-exploitation scheme well prevents overfitting of training process for sparse dataset.

Table 9: Ablation study on P with and without unsupervised loss term.

	Validation Score
Expert exploitation $\{P = 3\}$	12.97
+ Self- exploitation $\{\lambda = 8\}$	**12.98**
Expert exploitation $\{P = 5\}$	12.95
+ Self- exploitation $\{\lambda = 8\}$	**12.95**
Expert exploitation $\{P = 7\}$	12.93
+ Self- exploitation $\{\lambda = 8\}$	**12.95**

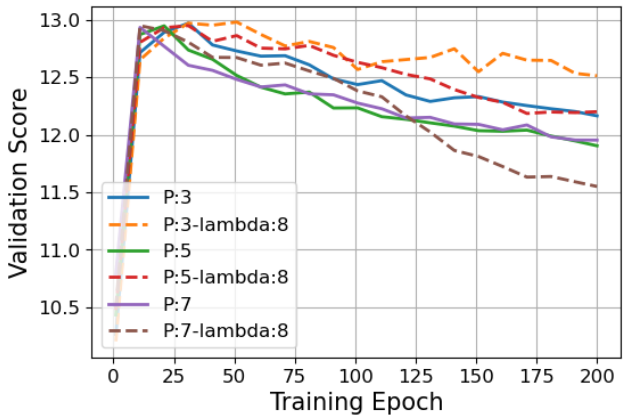


Figure 24: Validation score of P ablation with and without self-exploitation loss term.