ML-based Physical Design Parameter Optimization for 3D ICs: From Parameter Selection to Optimization

Hao-Hsiang Hsiao
thsiao@gatech.edu
Georgia Institute of Technology
Atlanta, GA, USA

Pruek Vanna-Iampikul
v.pruek@gatech.edu
Georgia Institute of Technology
Atlanta, GA, USA

Yi-Chen Lu
yclu@gatech.edu
Georgia Institute of Technology
Atlanta, GA, USA

Sung Kyu Lim
limsk@ece.gatech.edu
Georgia Institute of Technology
Atlanta, GA, USA

ABSTRACT

While various studies have shown effective parameter optimizations for specific designs, there is limited exploration of parameter optimization within the domain of 3D Integrated Circuits. We present the first comprehensive study, both qualitatively and quantitatively, comparing five state-of-the-art (SOTA) techniques for parameter optimization applied to 3D ICs. Additionally, we introduce an end-to-end machine learning-based framework, encompassing important parameter selection through optimization, all without human intervention. Extensive studies across six industrial designs under the TSMC 28nm technology node reveal that our proposed framework outperforms SOTA techniques in three different optimization objectives in both optimization quality and runtime.

ACM Reference Format:
Hao-Hsiang Hsiao, Pruek Vanna-Iampikul, Yi-Chen Lu, and Sung Kyu Lim. 2024. ML-based Physical Design Parameter Optimization for 3D ICs: From Parameter Selection to Optimization. In *61st ACM/IEEE Design Automation Conference (DAC '24), June 23–27, 2024, San Francisco, CA, USA.* https://doi.org/10.1145/3649329.3656509

1 INTRODUCTION

High-performance 3D ICs are commonly designed with pseudo-3D[1] approaches, which utilize existing 2D IC commercial to function as 3D IC CAD tools. These methods start with constructing a 2D physical design using commercial 2D IC physical design tools and then transform it into a 3D layout.

State-of-the-art (SOTA) pseudo-3D flows fall into two main categories: partitioning-first and partitioning-last. In partitioning-first flows, tier partitioning (TP) is conducted before global placement. In contrast, partitioning-last flows, like Compact-2D[2], perform TP after an intermediate placement stage. The Compact-2D flow emulates the final 3D IC footprint by scaling wire RC values by $1/\sqrt{2}$ without shrinking cells and wires to generate intermediate placement before TP. Pin-3D[3] is designed for true 3D routing and optimization using 2D tools after TP. It optimizes one tier iteratively, using transparent cells to represent the other tier's cells and projecting their pins to the metal stack. This approach, as demonstrated in [3], yields improved performance and lower power consumption compared to the original Compact-2D flow. However, the final design quality depends on the initial intermediate-2D design and tier partitioning result.

Although pseudo-3D flows often demonstrate superior Quality of Results (QoR) compared to their 2D counterparts or academic true-3D design flows, [4] shows that the consistent improvement of Power, Performance, and Area (PPA) metrics is not always guaranteed with pseudo-3D workflows. Pseudo-3D flows involve various parameters, including commercial tool parameters and tier-partitioning parameters, which significantly influence design quality. Therefore, to fully exploit the potential of pseudo-3D workflows, a properly optimized parameter configuration is crucial.

In this study, we explore parameter optimization for the Compact-2D + Pin-3D approach in Face-to-Face (F2F)-bonded 3D ICs. Despite considerable research in parameter autotuning within Electronic Design Automation (EDA), there is a notable gap in research dedicated to Design Space Exploration (DSE) in the context of 3D ICs. Our objective is to introduce a comprehensive end-to-end automatic tuning framework, covering parameter selection to optimization, and eliminating the need for human intervention. The contribution of our work is summarized as follows:

- We conducted the first comprehensive comparative study of current research in parameter optimization techniques within Electronic Design Automation (EDA) for 3D ICs.
- We introduce an end-to-end machine learning-based framework for seamless parameter selection and optimization in 3D ICs, eliminating the need for human intervention.
- We introduce a reinforcement learning (RL)-based parameter optimization framework that leverages advantages from existing approaches, consistently outperforming SOTA methods across various benchmarks and objectives.
- We introduce a hybrid framework that facilitates RL tuning using a reward estimator pretrained offline, eliminating the need for the time-consuming Pseudo-3D flow.

2 PRELIMINARIES

2.1 Parameter Selection

The Pseudo-3D flow provides a large spectrum of parameters, making it impractical to tune them blindly. Consequently, identifying the parameters that carry greater importance for our task becomes a crucial step. However, quantifying the importance of each parameter to the final PPA is often nontrivial and counter-intuitive; the interactions between parameters make this task even more challenging. Therefore, many statistical techniques are employed.

DAC '24, June 23–27, 2024, San Francisco, CA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0601-1/24/06.
https://doi.org/10.1145/3649329.3656509

Table 1: Selected Parameters.

ID	Parameter	type	range
1	place_glo_max_density	float	[0.6, 0.9]
2	place_glo_uniform_density	bool	2
3	place_glo_clock_power_driven	bool	2
4	place_glo_clock_power...effort	enum	3
5	place_det_wire_length_opt_effort	enum	3
6	place_glo_activity_power_driven	bool	2
7	place_glo_activity...effort	enum	2
8	place_glo_cong_effort	enum	5
9	place_glo_timing_effort	enum	2
10	max_fanout	float	[20, 100]
11	max_source_to_sink_net_length	float	[20, 200]
12	target_skew	float	[0.1, 0.24]
13	target_max_trans	float	[0.15, 0.85]
14	target_insertion_delay	float	[0, 0.12]
15	max_buffer_depth	int	[20, 100]
16	PdPUnbalance	float	[3, 15]
17	noPart	int	[20, 90]

Parameter selection methods can be broadly classified into testing-based and model-based approaches. Testing-based methods, such as Univariate Selection, rely on statistical tests to determine the significance of individual parameters. These methods offer convenience but overlook parameter interactions. On the other hand, model-based approaches consider parameter interactions but come with a higher cost. The process involves fitting a regression model that takes parameters as input and predicts the PPA. The importance of each parameter is based on its impact on the model output. In Tree-Based Selection, a decision tree recursively selects parameters to optimize dataset splitting, which inherently calculates an importance score for each parameter during the splits. Recursive Elimination (RE) and Sequential Selection (SS) are both model-based approaches that begin with all parameters and progressively remove the ones that least affect model performance upon removal, which can be measured by cross-validation or impurity scores.

2.2 Parameter Optimization

In this paper we implement and compare the following ML methods widely adopted for parameter optimization:

Bayesian Optimization (BO): Bayesian Optimization employs a surrogate Gaussian process model to probabilistically predict the PPA based on parameter configurations. With the prediction, an acquisition function guides the selection of the next configuration to sample. The chosen configuration undergoes evaluation with real 3D P&R, and the surrogate model is updated with the new observation. This iterative process enhances the accuracy of the surrogate model over successive steps.

Ant Colony Optimization (ACO): ACO, inspired by real ants, models parameter optimization as a graph. With m parameters, the graph has m+1 nodes, each representing a parameter. Edges between nodes, associated with pheromone levels, denote possible configurations for each parameter. Paths from the first to the last node correspond to parameter configurations. After evaluating performance, pheromone levels on the chosen path are updated based on QoR. This iterative process continues until convergence.

Algorithm 1: Our Parameter Selection Flow.

```
Input:
 1: $\mathbf{P}_{all} : \{p_1 \ldots, p_n\}$: a list of unselected parameters
Output: Parameters $\mathbf{P}_{select} \subset \mathbf{P}_{all}$ with high importance
 1: $P_{select} = \{\}$
 2: for each netlist $\mathcal{N}$ do
 3:     Run Pseudo-3D to collect a dataset $\mathcal{D}$ with randomly
        sampled of $\mathbf{P}_{all}$
 4:     for each optimization objective do
 5:         Train an XGBoost model $T$ with $\mathcal{D}$ that minimizes
            cross-validation RMSE score
 6:     end for
 7:     for each $p_i \in \mathbf{P}_{all}$ do
 8:         Calulate SHAP value $S_i$ for $p_i$ with $T$ and $\mathcal{D}$
 9:         if $S_i > threshold$ then
10:             $P_{select} = P_{select} \cup \{p_i\}$
11:         end if
12:     end for
13: end for
```

Recommendation-Based: [5] utilized matrix factorization (MF) for parameter suggestion. In our case, each netlist (p_i) and parameter configuration (q_j) have their latent features $\in \mathbf{R}^k$. The predicted QoR ($\hat{r_{ij}}$) for performing 3D P&R is given by $p_i \cdot q_j$. Latent features are trainable by minimizing $\sum_{i,j} |r_{ij} - \hat{r_{ij}}|$ for all completed netlist-parameter pairs. After training, these features predict unexplored pairs, suggesting configurations with high predicted QoR. However, [5] lacks leverage of domain knowledge, and may face challenges in high-dimensional or continuous design spaces.

Feature-Importance Sampling and Tree-Based (FIST): [6] proposed an efficient sampling strategy. They leverage the inherent characteristics of Tree-Based models, which recursively divide the dataset into subsets based on parameter importance. FIST employs a sampling methodology that selects parameter configurations within clusters after the dataset is split, facilitating an efficient exploration of various configurations.

Reinforcement Learning Based (RL): [7] proposed a RL-based tuning framework that iteratively adjusts a configuration proposed by a human expert, to optimize cumulative PPA rewards. However, the method starts from a preset configuration, introducing potential arbitrariness and uncertainty of reachability from a suboptimal starting point within a limited episode length. Since it doesn't predict the optimal parameter directly, this method aligns more closely with the parameter "fine-tuning."

3 OUR PARAMETER SELECTION

3.1 Challenges of Parameter Selection

Model-based selection algorithms are a preferable choice due to the interactions among parameters in P&R tools, involving a trade-off between performance and power. However, different methods and models with varying hyperparameters can yield diverse importance results that pose a significant challenge in identifying parameter importance. To address these issues, we propose a strategy to achieve consistent and reliable importance measurements.

3.2 Selection Methodology

Tree-based methods often suffer from inconsistency issues due to a lack of model performance evaluation, resulting in potentially biased assessments in cases of overfitting or biased datasets. To mitigate this, we incorporate cross-validation with Bayesian Optimization to train a reliable model structure for importance measurement. XGBoost stands out as an optimal selection due to its effectiveness and lightweight nature. With fewer hyperparameters, it produces more consistent results and a reduced search time.

We leverage SHAP (SHapley Additive exPlanations), which is a powerful tool for explaining the output of Machine Learning models, for importance measurement. SHAP excels in providing consistent and interpretable explanations, offering valuable insights into the influence of each parameter on the model's predictions.

Let $f : \mathcal{X} \rightarrow \mathbb{R}$ be a model that maps input instances x from the parameter space $\mathcal{X}$ to predictions. The SHAP value $\phi_i(x)$ for a specific parameter i in a given instance x is defined as follows:

$$\phi_i(x) = \sum_{S \subseteq \{1,\ldots,p\} \setminus \{i\}} \frac{|S|!(p - |S| - 1)!}{p!} [f(x_S \cup \{i\}) - f(x_S)] \tag{1}$$

where S is a subset of parameters excluding i, x_S represents the instance with only parameters in S, and p is the total number of parameters. The equation calculates the contribution of parameter i to the model prediction by considering all possible subsets of parameters. The term $f(x_S \cup \{i\}) - f(x_S)$ represents the change in the model prediction when parameter i is included.

The SHAP values satisfy the following desirable properties:

- **Local Accuracy:** $f(x) = \sum_i \phi_i(x)$ for all instances x.
- **Consistency:** If $x_i \leq x'_i \ \forall i$, then $\phi_i(x) \leq \phi_i(x') \ \forall i$.
- **Symmetry:** If $f(x) = f(x')$ for all instances x and x' that only differ in features i and j, then $\phi_i(x) = \phi_i(x')$ and $\phi_j(x) = \phi_j(x')$.

These properties ensure that SHAP values provide accurate and consistent measures of parameter importance, making them a powerful tool for parameter selection. Our overall selection mechanism is described in Algorithm 1.

3.3 Our Selected Parameters

Table 1 lists our selected parameters with Figure 1 visualizing the five most critical ones. Each subplot in the figure provides a summary of SHAP values for a parameter, where each dot represents a data point. The color of the dot corresponds to the parameter value. Absolute SHAP values indicate the impact on the model's output, with the sign representing positive or negative impact.

For example, Param. 11 significantly affects power consumption in both Rocket and AES designs by controlling the net length from the clock source to the sink, where a lower value increases power consumption. On the other hand, Param. 16, which governs the area unbalance for bin-based Tier-partitioning (TP), has the most significant influence on the WNS metric for Rocket. From Figure 1, within the specific range, higher values can enhance cut size, thereby avoiding the snaking of 3D nets.

The trade-off between timing and power often involves considering the PDP metric. In the case of AES, reducing Parameter 11 decreases PDP, whereas for Rocket, a reduction in Parameter 11 results in a higher PDP. This highlights that the optimization efforts during the P&R stage vary across different designs. As a result, 3D IC parameters should be specifically optimized for different designs to achieve the desired design goal.

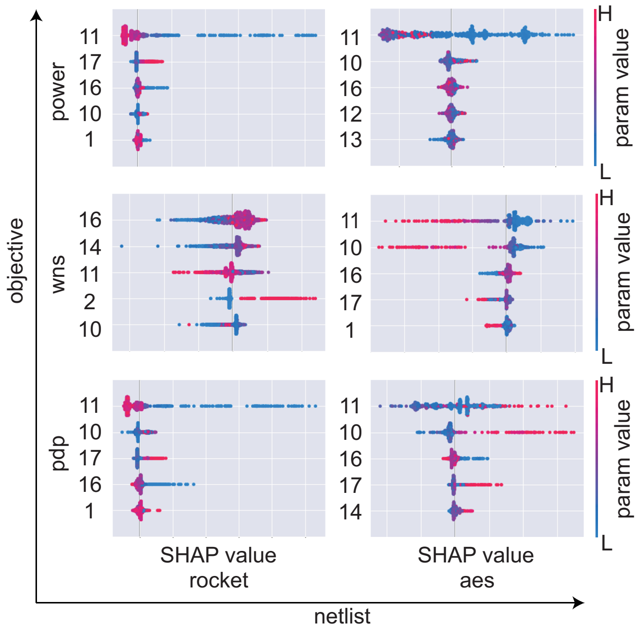

Figure 1: The important parameters of objective vs. netlist

4 OUR PARAMETER OPTIMIZATION

4.1 Motivation

Our first goal is to build a transferable framework that enables leveraging design features, which [7] successfully leveraged design features by training NN agent with RL. Our second goal is to reduce the runtime of online tuning. The advantage of online tuning is that it doesn't necessitate a large prebuilt dataset. However, the challenge is the slow turnaround time to obtain the PPA, which is the bottleneck of optimization. [5] utilizes offline supervised learning to predict the QoR for each design-parameter pair, enabling fast parameter suggestions without iteratively running P&R.

Therefore, we propose a novel tuning strategy that uses NN trained with RL to facilitate the utilization of design features. To shorten the long tuning process, we train an additional reward estimator to predict the QoR for each design-parameter pair.

4.2 Overall Optimization Flow

Figure 2 offers a high-level overview of our 3DTuner. We introduce a hybrid tuning framework that integrates both online and offline techniques. The tuning flow comprises the following phases:

(1) Offline Supervised Learning: We trained a reward estimator as a substitute for 3D P&R using supervised learning with a dataset $\mathcal{D}$ containing diverse netlist-configuration pairs, and their corresponding ground truth QoR. The training is to minimize the Mean Squared Error (MSE) loss between predicted and actual PPA values.

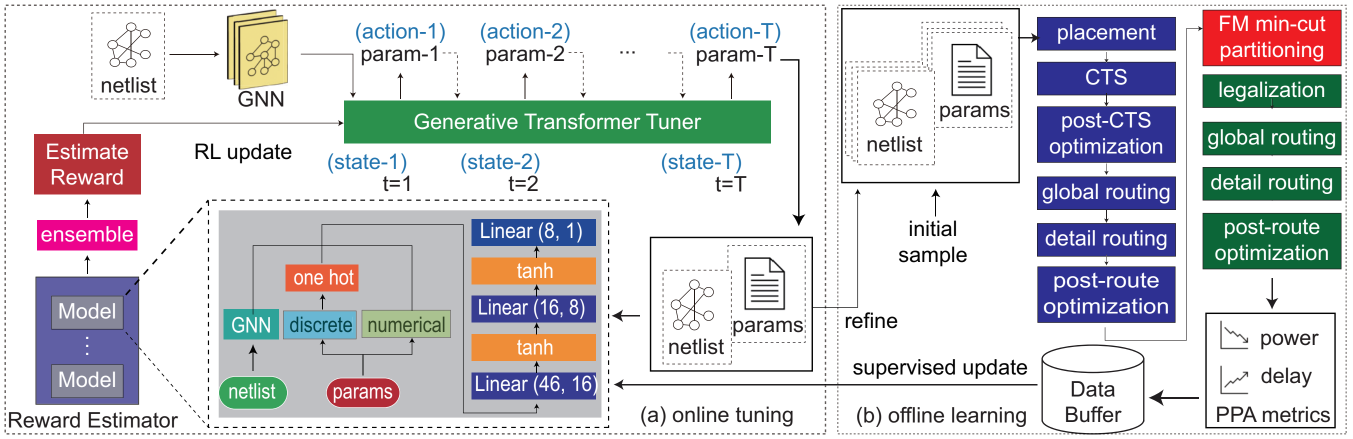

Figure 2: The 3DTuner sequentially optimizes parameters online to maximize the reward predicted by an offline pre-trained reward estimator. The estimator can undergo further refinement using online data by executing Pseudo-3D with the most promising configuration predicted by the 3DTuner.

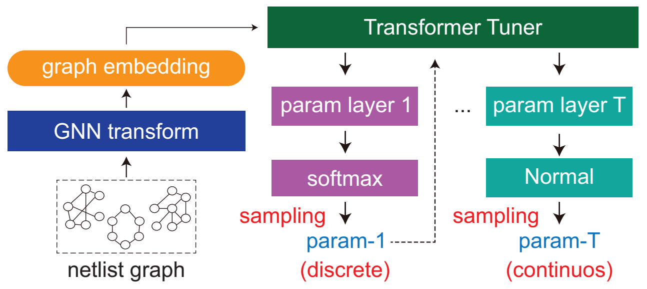

Figure 3: Our 3Dtuner includes: GNN, Transformer, and customized Parameter Layers

(2) Online tuning: the 3DTuner employs netlist embeddings and self-attention mechanism for autoregressive parameter tuning. It samples parameter actions at each time step, calculates rewards using the reward estimator, and updates its parameters with the PPO algorithm until the reward stabilizes.
(3) Online Refinement: To further improve our 3DTuner and reward estimator, after optimizing the estimated reward, the 3DTuner selects the most promising configuration to run through the real 3D P&R flow. The completed data is added to the dataset to refine the reward estimator.

4.3 Sequential Tuning with Transformer Tuner

We frame parameter optimization as a sequential decision-making process, where parameters $p_1, p_2, \ldots, p_T$ are tuned one after the other. This approach has two primary benefits. Firstly, it enables autoregressive tuning, where the tuning of each parameter is informed by the values for those before it. Secondly, it scales efficiently with the number of parameters, as it adds a decision step for each parameter rather than expanding the model's dimensionality. We chose the Transformer language model as our tuner due to its proficiency in handling sequential decisions. The self-attention mechanism in the Transformer automatically assesses the importance of previous outputs for making the current prediction. To account for the diverse range of parameters, we have introduced a dedicated parameter layer for each parameter following the shared Transformer layer. For discrete parameters, this layer is a softmax layer that calculates the probability distribution across possible options. For continuous parameters, the layer predicts the mean and standard deviation of a normal distribution, as depicted in Figure 3.

Table 2: Initial features of each node in our netlist graph.

features	descriptions
wst slack	worst slack of cell
wst output slew	maximum transition of output pin
wst input slew	maximum transition of input pin
drv net power	switching power of driving net
int power	cell internal power
leakage	cell leakage power

4.4 Netlist Encoding with GNN

We utilize GNN to train a transferable tuner, capturing both netlist connectivity and cell attributes. Node-level embedding begins with handcrafted features from cell metadata, as shown in Table 2. An aggregation function combines neighboring nodes' embeddings iteratively. Graph attention pooling retains high-attention nodes, creating a condensed graph embedding. A global mean aggregation consolidates node-level embeddings, forming a holistic graph representation. Our GNN framework has three graph convolution layers and a final fully-connected layer, all with a shared hidden dimension of 32, resulting in 16-dimensional graph embeddings.

4.5 Reinforcement Learning Formulation

To train with RL algorithm, we formulate the tuning process as a MDP. To allow direct prediction of parameters, our MDP settings differ from [7]. Below are our RL formulations:

- *Trajectory* (τ): A trajectory τ is the complete sequence of parameter selections from p_1 to p_T.
- *States* (s): At time step t, a state s_t includes the configuration of parameters tuned from time steps 1 through $t-1$, using the

Transformer's self-attention mechanism. It also integrates design characteristics encoded by GNN.

- *Actions* (a): An action a_t represents all potential configurations for the parameter to be tuned at time step t.
- *Reward* (r): our rewards are set to zero for intermediate actions $a_1, a_2, \ldots, a_{T-1}$, except for the last action, a_T, corresponding to the estimated PPA obtained after the 3D flow.

Each PPA estimator accepts input in the form of concatenated parameter embeddings and graph-extracted features, resulting in a total of 46 dimensions. This input is then processed through two hidden layers with 16 and 8 outputs, respectively. During training, the model is updated using the mean squared error (MSE) loss, calculated between the ground truth and the predicted PPA value.

3DTuner's learning depends heavily on the reward estimator, and relying on a single estimator is risky. To mitigate this, we incorporate estimation uncertainty by approximating Bayesian learning. By assuming uncertainty is parameterized by θ, the posterior distribution of θ is learned from data, and the expected reward is calculated as $\mathbf{E}[R] = \int R(\theta)p(\theta|\mathcal{D})d\theta \approx \frac{1}{N}\sum_i R(\theta_i)$. We approximate uncertainty by training an ensemble of 5 models.

4.6 Training Methodology

The overall process is illustrated in Algorithm 2. We update our 3DTuner to optimize the expected reward with the SOTA Proximal Policy Optimization (PPO) algorithm. The PPO clipped surrogate objective that we aim to maximize is as follows:

$$\mathcal{L}(\theta) = \mathbb{E}\left[\min\left(\rho(\theta)\hat{A},\ clip(\rho(\theta), 1-\epsilon, 1+\epsilon)\hat{A}\right)\right] \tag{2}$$

PPO promotes actions with positive advantage (outperforming expectations) and discourages actions with negative advantage (underperforming expectations). The advantage function $\hat{A}$ represents the difference between observed rewards R and expected returns.

$\rho(\theta)$ quantifies the likelihood of taking a current action under the new policy versus the old policy, governing policy updates. In clipped PPO, a key constraint using min and clip functions restricts policy updates to a specified range, typically $[1-\epsilon, 1+\epsilon]$ with ϵ conventionally set to 0.2. These constraints prevent overly large policy updates that could destabilize training.

5 EXPERIMENTAL RESULTS

We conduct experiments on six industrial benchmarks, as listed in Table 3, to assess the optimization capabilities of SOTA algorithms across three different objectives: WNS, total power, and PDP. We employ Cadence Innovus for Cascade-2D + Pin-3D using face-to-face hybrid bonding with 1μm pitch under 28nm technology nodes with parameter selection and optimization implemented in Python.

5.1 Experiment Setup

To evaluate the effectiveness of the existing optimization algorithms, we implemented the following:

Algorithm 2: Our Parameter Optimization Flow.

```
Input:
1: $\mathbf{P}_{select}: \{p_1 \ldots, p_n\}$: all parameters to be tuned
2: GNN for gate-level netlists
3: Dataset $\mathcal{D} = \{(P_1, G_1, y_1), \ldots (P_N, G_N, y_N)\}$ which contains
   parameter-netlist pairs and their ground truth PPAs
Output: parameters $\mathbf{P}^*: \{p_1^* \ldots p_n^*\}$ that optimize PPA
1: Initialize the Reward_Estimator with weight $\theta_{Est}$
2: Initialize the 3DTuner with weights $\theta_{Tuner}$
3: repeat
4:     repeat
5:         $\mathbf{y}' \leftarrow Reward\_Estimator(\mathbf{P}, \mathbf{G})$
6:         Update $\theta_{Est}$ to minimize MSE loss: $J(\theta) \leftarrow \frac{1}{2m}\|\mathbf{y} - \mathbf{y}'\|_2^2$
7:     until $\theta_{Est}$ converged
8:     repeat
9:         $\mathbf{g} \leftarrow GNN(G)$
10:        $p_i' \sim 3DTuner(p_i'|p_1', \ldots, p_{i-1}', g)$
11:        Compute reward: $r \leftarrow Reward\_Estimator(p_1', \ldots, p_n', g)$
12:        Update $\theta_{Tuner}$ by maximizing Eq. 2 using gradient ascent
13:    until reward saturated
14:    Query most promising $\hat{P}^* \sim 3DTuner$
15:    Run Pseudo-3D with $\hat{P}^*$, obtained ground truth y
16:    $\mathcal{D} = \mathcal{D} \cup \{(\hat{P}^*, G, y)\}$
17: until objective is optimized
```

(1) **BO:** Similar setting as in [9]. We implement an asynchronous setting with shared data. Matern Kernel is used and different acquisitions (EI, UCB, POI) are used for different workers. Offline data is used as prior for the same design.
(2) **ACO:** Similar settings as [8]. Three ACO engines are employed for different groups of parameters, and continuous parameters are discretized.
(3) **Recommender:** Similar as [5], we implement MF algorithm with offline data, and continuous parameters are discretized.
(4) **FIST:** We implement a similar approach based on [6] using XGBoost. Offline data is used for model construction.
(5) **RL-based:** We implement the similar MDP setting as in [7] using StableBaseline and OpenAI Gym API. The episode length is set to 16 as described in [7].

We begin by sampling 1500 data points with randomly generated configurations across six tuning designs, forming an offline dataset that serves as prior knowledge for ACO, BO, and FIST. It also acts as training data for our reward estimators and the latent features of the Recommender. Notably, BO is restricted to data from the same netlist, and RL[7] lacks a direct method to leverage offline data.

Within our framework, we conduct experiments to assess results with and without online refinement. In the case of online refinement, we allocate a tool run budget of 20 for each design. This process not only optimizes the reward estimated by the reward estimator but also involves running Pseudo-3D with promising parameters to refine the reward estimator based on the newly acquired experience.

5.2 Optimization Result

The results in Table 3 indicate our framework consistently surpasses SOTA methods. It utilizes offline data for transfer, outperforming those with limited transferability. Our online refinement further enhances performance but at the cost of online tool runs.

Despite the Recommender using neural networks for transferability, its reliance on collaborative filtering, overlooking known features, and restricting parameter solution granularity, leads to suboptimal results compared to SOTA. RL[7] emerges as a promising approach; however, it exhibits sensitivity to the starting point.

Table 3: Comparison of tuning results between ours and SOTA [4 – 8] methods in optimizing 3 different objectives. Power, worst negative slack (WNS), and power-delay product (PDP) are reported in mW, ps, and pJ, respectively. 'Tool' represents the automatic setting of the commercial tool and the tier-partitioner. The values in parentheses denote the percentage change compared to the 'tool' setting.

Metrics	tool	ACO[8]	BO[9]	Recommend[5]	FIST[6]	RL[7] (bad start)	RL[7]	Ours	Ours (refine)
AES (#cells: 112K, #nets: 112K, #IO: 390)									
power	433.44	423.62 (-2.26%)	423.54 (-2.28%)	426.33 (-1.64%)	422.18 (-2.59%)	445.22 (+2.72%)	416.4 (-3.93%)	420.23 (-3.05%)	416.35 (-3.94%)
wns	-50	-25 (-50.00%)	-27 (-46.00%)	-33 (-34.00%)	-28 (-44.00%)	-67 (34.00%)	-20 (-60.00%)	-24 (-52.00%)	-22 (-56.00%)
pdp	115.32	108.33 (-6.06%)	106.1 (-8.00%)	107.53 (-6.76%)	105.61 (-8.42%)	124.07 (7.59%)	104.45 (-9.43%)	106.24 (-7.87%)	103.53 (-10.22%)
Rocket Core (#cells: 120K, #nets: 120K, #IO: 379)									
power	177	172.64 (-2.46%)	172.62 (-2.47%)	173.45 (-2.01%)	172.26 (-2.68%)	181.27 (2.41%)	171.88 (-2.89%)	172.21 (-2.71%)	171.02 (-3.38%)
wns	-32	-12 (-62.50%)	-6 (-81.25%)	-14 (-56.25%)	-9 (-71.88%)	-35 (9.38%)	-5 (-84.38%)	-4 (-87.50%)	-2 (-93.75%)
pdp	182.66	176.76 (-3.23%)	175.72 (-3.80%)	176.98 (-3.11%)	176.17 (-3.55%)	180.94 (-0.94%)	172.04 (-5.81%)	174.54 (-4.45%)	171.46 (-6.13%)
ECG (#cells: 83K, #nets: 84K, #IO: 1.7K)									
power	534.63	518.02 (-3.11%)	514.67 (-3.73%)	522.6 (-2.25%)	517.04 (-3.29%)	548.75 (2.64%)	513.05 (-4.04%)	515.22 (-3.63%)	512.98 (-4.05%)
wns	-84	-57 (-32.14%)	-58 (-30.95%)	-67 (-20.24%)	-52 (-38.10%)	-101 (20.24%)	-53 (-36.90%)	-52 (-38.10%)	-49 (-41.67%)
pdp	161.24	154.57 (-4.14%)	150.5 (-6.66%)	155.4 (-3.62%)	153.28 (-4.94%)	180.54 (11.97%)	146.27 (-9.28%)	148.8 (-7.72%)	142.92 (-11.36%)
DMA (#cells: 13K, #nets: 14K, #IO: 961)									
power	66.52	63.02 (-5.26%)	63.48 (-4.57%)	63.77 (-4.13%)	63.81 (-4.07%)	66.7 (+0.27%)	61.49 (-7.56%)	63.55 (-4.46%)	62.28 (-6.37%)
wns	-120	-91 (-24.17%)	-94 (-21.67%)	-101 (-15.83%)	-92 (-23.33%)	-121 (0.83%)	-91 (-24.17%)	-95 (-20.83%)	-88 (-26.67%)
pdp	23.09	21.97 (-4.85%)	21.95 (-4.94%)	21.94 (-4.98%)	20.76 (-10.09%)	23.89 (3.46%)	20.81 (-9.87%)	21.86 (-5.33%)	20.06 (-13.12%)
VGA (#cells: 52K, #nets: 52K, #IO: 184)									
power	240.6	235.99 (-1.92%)	235.62 (-2.07%)	235.4 (-2.16%)	233.78 (-2.83%)	244.03 (1.43%)	232.43 (-3.40%)	234.3 (-2.62%)	232.33 (-3.44%)
wns	-125	-51 (-59.20%)	-47 (-62.40%)	-46 (-63.20%)	-50 (-60.00%)	-142 (13.60%)	-40 (-68.00%)	-45 (-64.00%)	-43 (-65.60%)
pdp	113.78	99.58 (-12.48%)	99.31 (-12.72%)	102.35 (-10.05%)	98.53 (-13.40%)	118.83 (4.44%)	94.35 (-17.08%)	99.02 (-12.97%)	93.16 (-18.12%)
LDPC (#cells: 39K, #nets: 41K, #IO: 4.1K)									
power	203.12	199.59 (-1.74%)	199.3 (-1.88%)	200.58 (-1.25%)	198.68 (-2.19%)	207.96 (2.38%)	196.44 (-3.29%)	198.56 (-2.24%)	195.88 (-3.56%)
wns (ps)	-29	-5 (-82.76%)	-7 (-75.86%)	-5 (-82.76%)	-8 (-72.41%)	-30 (3.45%)	-2 (-93.10%)	-5 (-82.76%)	-3 (-89.66%)
pdp	107.45	100.07 (-6.87%)	102.69 (-4.43%)	99.94 (-6.99%)	102.48 (-4.63%)	108.68 (1.14%)	99.01 (-7.85%)	101.32 (-5.70%)	99.35 (-7.54%)
Total number of tool runs for each objective (6 designs)									
# run	6	300	300	6	240	120	120	6	120

In addition to the normal setting, we adversarially sample an extremely poor starting point based on offline data. We observed that the agent struggled to rectify when the starting point was too unfavorable. Yet, when provided with an oracle of a good starting point, this method proves effective as a practical fine-tuning framework.

6 CONCLUSION

We conducted the first comparative study on parameter optimization for 3D ICs, introducing an end-to-end framework that spans from parameter selection to optimization, thereby eliminating the need for human intervention. Extensive experiments show that our framework, leveraging transfer learning via GNN, achieves high transferability across diverse designs. Our methods outperform state-of-the-art approaches within seconds and are capable of further refinement online to achieve even more optimal results.

ACKNOWLEDGMENTS

This work was supported by the National Science Foundation and the industry members of the CAEML I/UCRC and Samsung Advanced Institute of Technology (SAIT) under the AI for Semiconductors Program.

REFERENCES

[1] Heechun Park et al. Pseudo-3d physical design flow for monolithic 3d ics: Comparisons and enhancements. ACM Trans. Des. Autom. Electron. Syst., 26(5), jun 2021.

[2] Bon Woong Ku et al. Compact-2D: A Physical Design Methodology to Build Two-Tier Gate-Level 3-D ICs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 39(6):1151–1164, 2020.

[3] Sai Surya Kiran Pentapati et al. Pin-3D: A Physical Synthesis and Post-Layout Optimization Flow for Heterogeneous Monolithic 3D ICs. In 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD), pages 1–9, 2020.

[4] Gauthaman Murali et al. ART-3D: Analytical 3D Placement with Reinforced Parameter Tuning for Monolithic 3D ICs. In Proceedings of the 2022 International Symposium on Physical Design, ISPD '22, page 97–104, New York, NY, USA, 2022. Association for Computing Machinery.

[5] Jihye Kwon, Matthew M. Ziegler, and Luca P. Carloni. A Learning-Based Recommender System for Autotuning Design Flows of Industrial High-Performance Processors. In Proceedings of the 56th Annual Design Automation Conference 2019, DAC '19, New York, NY, USA, 2019. Association for Computing Machinery.

[6] Zhiyao Xie, Guan-Qi Fang, Yu-Hung Huang, Haoxing Ren, Yanqing Zhang, Brucek Khailany, Shao-Yun Fang, Jiang Hu, Yiran Chen, and Erick Carvajal Barboza. FIST: A Feature-Importance Sampling and Tree-Based Method for Automatic Design Flow Parameter Tuning. In 2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC), pages 19–25, 2020.

[7] Anthony Agnesina et al. VLSI Placement Parameter Optimization using Deep Reinforcement Learning. In 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD), pages 1–9, 2020.

[8] Rongjian Liang et al. FlowTuner: A Multi-Stage EDA Flow Tuner Exploiting Parameter Knowledge Transfer. In 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD), page 1–9. IEEE Press, 2021.

[9] Yuzhe Ma et al. CAD Tool Design Space Exploration via Bayesian Optimization. In 2019 ACM/IEEE 1st Workshop on Machine Learning for CAD (MLCAD), pages 1–6, 2019.