# Rank-DSE: Neural Pareto Comparator of Microarchitecture Design Space Exploration

PENG XU, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, Hong Kong
SU ZHENG, Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, Hong Kong
MINGZI WANG, Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, Hong Kong
ZIYANG YU, The Chinese University of Hong Kong, Hong Kong, Hong Kong
SHIXIN CHEN, The Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, Hong Kong
TINGHUAN CHEN, The Chinese University of Hong Kong - Shenzhen, Shenzhen, China
KEREN ZHU, Fudan University, Shanghai, China
TSUNGYI HO, The Chinese University of Hong Kong, Hong Kong, Hong Kong
BEI YU, CSE, The Chinese University of Hong Kong, Hong Kong, Hong Kong

The complexity of microarchitecture design has surged due to the expanding design space and time-intensive verification processes. Existing regression-based machine learning methods struggle with inaccurate estimations because of limited training samples. To address these challenges, we propose Rank-DSE, a novel framework for microarchitecture design space exploration (DSE) that leverages a Neural Pareto Comparator (NPC) to directly model the comparative relationships between different architecture designs. Rank-DSE bypasses the inaccuracies of absolute PPA (performance, power, area) predictions by focusing on relative comparisons. The NPC computes the probability of one architecture dominating another and employs semi-supervised learning to reduce the reliance on labeled data. Additionally, a reinforcement-learning-based sampling scheme with an updating baseline Pareto set accelerates the exploration process. Experimental results on the ICCAD 2021 benchmark demonstrate that Rank-DSE achieves superior search quality and

Peng Xu and Su Zheng equally contributed to this research.

This work is partially supported by The Research Grants Council of Hong Kong SAR (No. RFS2425-4S02, CUHK14201624 and CUHK14210723), The National Natural Science Foundation of China (No. 62304197), the State Key Laboratory of Integrated Chips and Systems, Fudan University (No. SKLICS-Z202404), and the Science and Technology Commission of Shanghai Municipality Project (24JD1400800).

Authors' Contact Information: Peng Xu, Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, Hong Kong; e-mail: pxu22@cse.cuhk.edu.hk; Su Zheng, Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, Hong Kong; e-mail: szheng22@cse.cuhk.edu.hk; Mingzi Wang, Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, Hong Kong; e-mail: wmzmaxwmzmax@gmail.com; Ziyang Yu, The Chinese University of Hong Kong, Hong Kong, Hong Kong; e-mail: zyyu21@cse.cuhk.edu.hk; Shixin Chen, The Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, Hong Kong; e-mail: sxchen@link.cuhk.edu.hk; Tinghuan Chen, The Chinese University of Hong Kong—Shenzhen, Shenzhen, Guangdong, China; e-mail: chentinghuan@cuhk.edu.cn; Keren Zhu, Fudan University, Shanghai, China; e-mail: krzhu@fudan.edu.cn; Tsungyi Ho, The Chinese University of Hong Kong, Hong Kong, Hong Kong; e-mail: tyho@cse.cuhk.edu.hk; Bei Yu (corresponding author), CSE, The Chinese University of Hong Kong, Hong Kong, Hong Kong; e-mail: byu@cse.cuhk.edu.hk.


ACM 1084-4309/2025/08-ART71
https://doi.org/10.1145/3747294

cost-efficiency compared to state-of-the-art methods. Specifically, Rank-DSE improves hypervolume by up to 7% while reducing exploration cost by 53.09% compared to cutting-edge approaches. These results highlight the advantages of Rank-DSE in terms of efficiency and effectiveness for microarchitecture DSE.

CCS Concepts: • **Computer systems organization**; • **Computing methodologies** → **Machine learning approaches**;

Additional Key Words and Phrases: Microprocessor, microarchitecture, design space exploration

**ACM Reference Format:**
Peng Xu, Su Zheng, Mingzi Wang, Ziyang Yu, Shixin Chen, Tinghuan Chen, Keren Zhu, Tsungyi Ho, and Bei Yu. 2025. Rank-DSE: Neural Pareto Comparator of Microarchitecture Design Space Exploration. *ACM Trans. Des. Autom. Electron. Syst.* 30, 5, Article 71 (August 2025), 24 pages. https://doi.org/10.1145/3747294

## 1 Introduction

The process of microprocessor design demands substantial human efforts. It involves optimizing **performance, power, and area** (**PPA**) through microarchitecture. Identifying an efficient PPA tradeoff early can reduce costs and the design cycle. However, increasing computational requirements and shrinking feature sizes complicate microarchitecture. A typical industrial approach involves manual configurations of design parameters by computer architects, which necessitates domain expertise and substantial labor. Additionally, the growing design spaces pose challenges due to their vast size (e.g., the RISC-V BOOM design space can reach $1.6 \times 10^8$ [1]). More commonly utilized solutions employ black-box methods [2–6]. However, complex feature interactions and mixed-type parameters also render statistical methods ineffective [7, 8].

To tackle these challenges, machine learning approaches have been introduced for efficient modeling of processor microarchitectures [8–10]. These methods typically treat the **design space exploration** (**DSE**) as a regression problem and employ surrogate regression models, such as **linear regression** (**LR**), **support vector machine** (**SVM**), and **Gaussian process** (**GP**) [8–10]. These models are trained using sampled datasets generated by processor simulations. Once trained, these surrogate models enable rapid prediction of PPA metrics, allowing thousands of predictions within seconds. Architects can effectively identify the most promising next steps by evaluating the predicted PPA metrics of different architecture configurations.

Although earlier machine learning approaches have shown promising results, they primarily focus on predicting the **exact** PPA metrics of a specific design space. However, using absolute prediction suffers from two limitations. First, achieving optimal prediction performance is challenging due to over-fitting and limited sample sizes, resulting in unstable predictions. Second, the regression-based DSE often shows a sloppy sampling estimation due to the unreliable prediction of expected improvement. Researchers attempt to mitigate the instability of regression-based models by incorporating prior knowledge, such as active learning for initial samples [10] or embedding relationships between features [11]. As shown in Figure 1, regression-based methods, despite having lower **mean square error** (**MSE**) loss, are likely to make incorrect estimates (black circles) when estimating Pareto boundaries due to ignoring the ranking relationship between configurations. Such erroneous estimates can lead to biased guidance when exploring new configurations, resulting in poor quality of the final solution.

The limitations inherent in PPA prediction underscore the need to consider the following question: *Is predicting the absolute PPA metrics necessary?* Or, can we employ a relative approach for evaluating architecture candidates instead? In the DSE problem, the primary objective is to identify designs that can achieve a Pareto front for PPA, which consists of the dominating designs. The key aspect

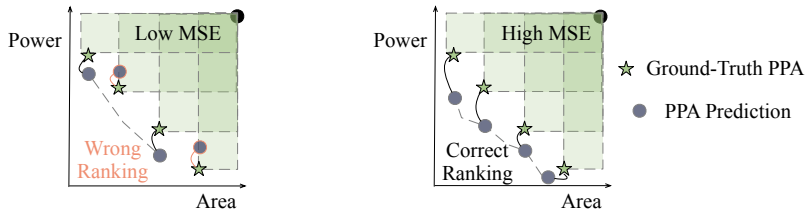


Fig. 1. Regression-based method with low MSE loss and wrong ranking (upper) vs. ranking-based method with high MSE loss and correct ranking (lower).

is to determine whether a point is preferred over others, which is equivalent to establishing their comparative relationships in PPA metrics.

By adopting a ranking-based approach, we can directly model these comparative relationships. Previous research has recognized the benefits of the ranking-based paradigm in DSE problems [12] but lacks comprehensive consideration of PPA metrics and effective sampling during inference time, resulting in limited efficiency. Recently, **neural architecture search** (**NAS**), as a subproblem of DSE, has experienced a resurgence by incorporating advancements in ranking techniques to improve search quality [13, 14]. Leveraging these advancements in ranking techniques presents an exciting opportunity to revisit the ranking-based paradigm in the microarchitecture DSE problem.

In this article, we propose to formulate the microarchitecture DSE problem as a ranking problem by leveraging the **Neural Pareto Comparator** (**NPC**), called **ranking-based design space exploration** (**Rank-DSE**). In summary, our main contributions are

- We present a ranking-based algorithm for microarchitecture DSE, namely, Rank-DSE, directly modeling comparison relationships to circumvent the distorted predictions from regression.
- Our NPC computes the dominating probability between microarchitecture designs, and uses a semi-supervised way to further enhance its generalizability.
- To further refine the quality of compared samples, we utilize a **reinforcement learning** (**RL**) sampling method with an updating baseline Pareto set.
- Compared with state-of-the-art baselines, our Rank-DSE method shows superior performance in both the quality and cost of search results. Especially, Rank-DSE brings up to 7% improvement in **hypervolume** (**HV**) and takes only 53.09% exploration cost compared with the cutting-edge methods, and outperforms the winners of the ICCAD CAD Contest 2022.

The overall structure of our article is organized as follows. Section 2 introduces the preliminaries. Section 3 shows the details of the proposed method. Section 4 presents experimental results that can prove the effectiveness of the RankDSE framework. Section 5 provides a conclusion to our article.

## 2 Preliminaries

### 2.1 Typical Microprocessor Components

The **Berkeley out-of-order machine** (**BOOM**) processor, a superscalar RISC-V architecture, has attracted attention for its competitive PPA metrics in low-power embedded cores. This article aims to optimize key components in BOOM's microarchitecture design, as depicted in Table 1.

The microarchitecture pipeline consists of key components contributing to efficient instruction execution and enhanced microprocessor performance. In the front end, the **instruction fetch unit** (**IFU**) utilizes the L1 instruction cache for fast instruction fetches. Decoded instructions are then processed by the **instruction decode unit** (**IDU**) to enable out-of-order execution and alleviate data dependency constraints. In the back end, the integer execution unit (INT) handles

integer and bitwise operations, while the **floating-point unit** (**FPU**) performs floating-point functions, including trigonometric and logarithmic tasks. The **load/store unit** (**LSU**) manages data movements between the function unit and the L1 data cache, ensuring quick access to frequently used data. Additionally, the larger L2 cache optimizes data retrieval. Together, these components enable efficient instruction execution and enhance microprocessor performance.

### 2.2 Learning to Rank (L2R)

L2R is originally an important research direction in the fields of information retrieval and machine learning, focusing primarily on how to use machine learning technology to rank search results [15, 16] effectively. L2R technology is commonly applied in various scenarios, such as search engine optimization and recommendation systems, to improve user satisfaction and system efficiency.

The common L2R techniques can mainly be divided into three strategies:

— Pointwise: Each query-document pair is treated as an independent sample, typically converted into a classification or regression problem [17, 18].
— Pairwise: The relative order between document pairs returned by a query is considered, transforming the problem into a binary classification issue [19–21].
— Listwise: The list of documents for each query is regarded as a whole, optimizing the ranking quality of the entire list [22–25].

### 2.3 DSE

Numerous algorithms have been introduced to address the microprocessor DSE problem [10, 17, 26–28], and they can be divided into analytical and black-box approaches. Analytical methods necessitate significant expert input to develop interpretable equations or micro-execution graph models that represent micro-architecture performance [26, 27]. These equations or graph models are then utilized to reduce the design space or facilitate rapid DSE. Karkhanis and Smith created analytical models to investigate an out-of-order superscalar microprocessor [26], while Golestani et al. [27] improved the dynamic event-dependence graph for DSE. When domain knowledge is unavailable, black-box methods are deployed to tackle the challenge [10, 17, 28]. Ipek et al. [28] utilize **artificial neural networks** (**ANNs**), and Li et al. [17] apply statistical sampling with black-box models like AdaBoost for DSE. Bai et al. [10] apply the GP model with Bayesian optimization to optimize the architecture configurations.

ML-based methods have also been proposed to address the DSE problems in different processor systems. For instance, Qian et al. [29] introduced a predictive model using **Support Vector Regression** (**SVR**) to optimize the microarchitecture settings of **network–on–chip** (**NoC**) architectures [30]. Patra et al. further explore regression models, including KNN, decision trees, and random forest, in the performance assessment of network–on–chip architectures. Regarding heterogeneous NoC architectures, Xiao et al. [31] proposed an end-to-end machine learning framework to optimize architectural parameters [32] using regression-based predictors, including LR, SVM, and **Naive Bayes** (**NB**). GNNs are further utilized in their following works [33, 34] to improve the prediction performance in architectural parameters of heterogeneous NoC architectures design. Their works have made significant contributions to the field of heterogeneous computing systems.

### 2.4 Problem Formulation

In this article, we attempt to explore and optimize the Pareto front for the BOOM microarchitecture design space.

*Definition 1 (Pareto Optimality).* For a multi-objective minimization problem with $M$ objectives, a solution $\boldsymbol{x}_1$ is deemed to dominate solution $\boldsymbol{x}_2$ if, for all $m$ belonging to the set $\{1, ..., M\}$, the

Table 1. Microarchitecture Design Space of BOOM [10]

| Modules | Nodes | Parameters | Values |
|---|---|---|---|
| Front-End | IFU | FetchWidth | 4,8 |
| | | FetchBufferEntry | 8,16,24,32,35,40 |
| | | FetchTagetQueueEntry | 16,40,32 |
| | | BranchCount | 8,6,10,12,10,14,16 |
| | | ICacheSets | 32,64,128 |
| | | ICacheWays | 4,8 |
| | | ICacheTLBSets | 1,2 |
| | | ICacheTLBWays | 8,16,32 |
| | IDU | DecodeWidth | 14,18,20,22,24,20,26 |
| | | RobEntries | 1,2,3,4,5 |
| | PRF | Int/FpPhysRegisters | 32,64,96,128,130 |
| Execute-Core | INT | IntDispatchWidth | 1,64,80,96,112 |
| | | IntNumEntries | 4,8,12,16,20,24,32,40 |
| | | IntIssueWidth | 1,2,3,4,5 |
| | FP | FpDispatchWidth | 1,64,80,96,112 |
| | | FpNumEntries | 4,8,12,16,20,24,32,40 |
| | | FpIssueWidth | 1,2 |
| | MEM | MemDispatchWidth | 1,64,80,96,112 |
| | | MemNumEntries | 4,8,12,16,20,24,32,40 |
| | | MemIssueWidth | 1,2 |
| | LSU | Ld/StqEntries | 8,16,24,32 |
| | | DCacheSets | 32,64,128 |
| | | DCacheWays | 2,4,8 |
| | | DCacheTLBSets | 1,2 |
| | | DCacheTLBWays | 8,16,32 |
| | | DCacheMSHR | 2,4,8 |
| | | ReplacementPolicy | 0,1 |

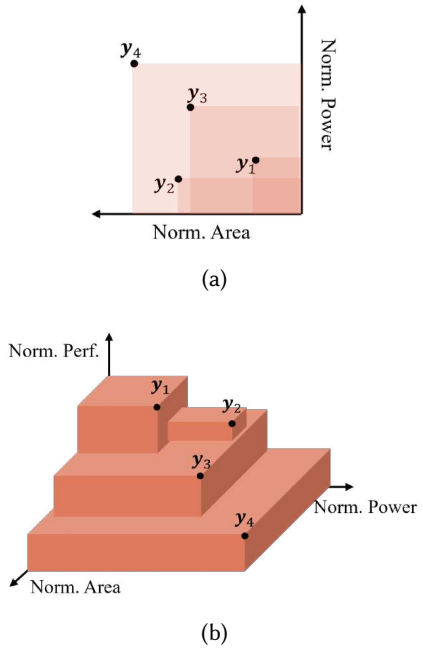


Fig. 2. (a) An example overview of the Pareto HV in the two-dimensional space; (b) An example overview of the Pareto HV in the three-dimensional space .i.e., power, performance, and area.

inequality of objective vectors $f_m(\boldsymbol{x}_1) \leqslant f_m(\boldsymbol{x}_2)$ holds true, and there exists at least one $k$ within the same set such that $f_k(\boldsymbol{x}_1) < f_k(\boldsymbol{x}_2)$; this relationship is symbolized by $\boldsymbol{x}_1 \succeq \boldsymbol{x}_2$. The collection of solutions that remain non-dominated by others constitutes the Pareto-optimal set, thereby establishing Pareto optimality within the design space. This collection of Pareto-optimal solutions, which represents the optimal balance of competing objectives, is referred to as the Pareto front.

As shown in Figure 2(a), when considering only the two metrics of Power and Area, $y_2$ and $y_3$ Pareto dominate $y_4$, which can be expressed as $y_2 \succeq y_4, y_3 \succeq y_4$. $y_1$, in turn, Pareto dominates $y_2$, $y_3$, and $y_4$. For the three-dimensional case, Figure 2(b) illustrates the Pareto dominance relationships when three metrics are included, where $y_1$ and $y_2$ Pareto dominate $y_3$ and $y_4$. Moreover, there is no Pareto dominance relationship between $y_1$ and $y_2$.

Problem 1 (Microarchitecture DSE). *Given a microarchitecture design space $\mathcal{D}$, each microarchitecture configuration inside $\mathcal{D}$ is regarded as a feature vector $x$ and the corresponding PPA metrics $y$ form the objective space $\mathcal{Y}$. For each $x$, the corresponding PPA metrics $y \in \mathcal{Y}$ can be estimated through the VLSI implementation flow $f$. Microarchitecture DSE is to explore the Pareto optimality, with the goal of minimizing exploration cost and runtime.*

## 3 Algorithm

This section introduces the proposed Rank-DSE framework and provides a detailed explanation of the algorithm. Our algorithm distinguishes itself from regression-based DSE methods that focus on predicting absolute PPA values. Instead, we employ a comparator model to predict the relevant relationship between architectures and leverage it to select the dominating Pareto front

set. However, there are several major algorithmic challenges in enabling this ranking-based DSE methodology.

(1) **Multi-objective Ranking Problem:** The existing ranking models typically utilize ranking information for a single metric [12], while the DSE problem entails finding a tradeoff among multiple objectives. In Rank-DSE, we propose a comparator model that compares any two architectures to provide the dominating probability between them.
(2) **Data-efficient Training Scheme:** Constructing accurate models for DSE is challenging due to the difficulty in obtaining sufficient data. In Rank-DSE, we further develop a semi-supervised learning scheme to exploit unlabeled architecture samples to improve generalizability.
(3) **Efficient Sampling with RL:** The computational cost of performing non-dominated sorting over the entire design space can be significant. To address this issue, we employ RL with an updating baseline set to enable efficient sampling during the inference stage.

In the rest of this section, we introduce the graph modeling of micro-architecture (Section 3.2), our NPC (Section 3.3), the semi-supervised learning scheme (Section 3.4), the RL aided sampling strategy (Section 3.5), the detailed algorithm (Section 3.6), and the complexity comparison between different methods (Section 3.7).

### 3.1 Overview

The overall exploration flow of our Rank-DSE framework is depicted in Figure 3(a). The process begins by creating an initial model using a set of starting designs and their associated PPA (Power, Performance, Area) metrics obtained from the VLSI flow. It then explores the solution space by sampling new designs based on this model. The PPA metrics of these sampled designs are determined by the VLSI flow. The model can be continuously refined by incorporating feedback from designs that have been evaluated, including the initial ones. Once the algorithm's stopping criteria are met, the Pareto-optimal microarchitectures are identified from the pool of explored designs.

We further detailed the model training and sampling part in the Rank-DSE as in Figure 3(b). To reduce the comparison cost, we sample several new designs using an RL policy via the average comparison probability predicted by NPC. Then, we update the baseline Pareto front set with the new designs. Finally, we evaluate the selected designs and determine if the termination condition is satisfied. In each iteration, an NPC model is trained to predict the dominating probability between two **directed acyclic graphs** (**DAGs**) of microarchitectures. A semi-supervised learning method for NPC is then employed to improve the generalizability further.

### 3.2 Graph Modeling for Micro-architecture

As shown in the left part of Figure 4(a), the BOOM processor's core pipeline is divided into two main stages: (1) Front End; (2) Execution End. In the front-end stage, there are fetch and decode Units, while in the execution stage, there are integer, floating-point, memory execution Units, and physical register units. Instructions are fetched and decoded in the front-end stage. These decoded instructions are then sent to the Execute stage for processing. Finally, the execution results are passed to the LSU for storage. Due to the processor's inherent pipeline structure, there are certain data dependencies among its various components. For example, in the Front End stage, the instruction cache, fetch unit, and fetch buffer supply fetched instructions to the decode unit for decoding. Data from the physical register is passed to the integer, floating-point, and memory units for execution. This introduces micro-architecture dependencies between downstream and upstream components, which are represented by micro-architecture parameters. The interdependence between components can be utilized to facilitate the microarchitecture optimization process, according to [11].

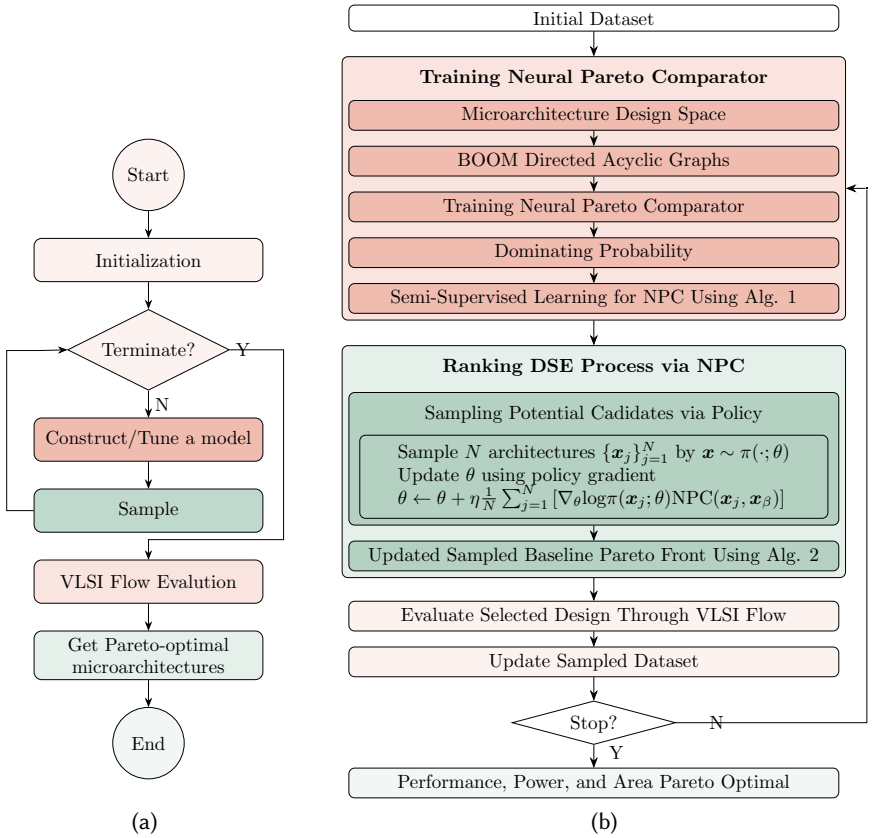


Fig. 3. The general exploration flow and detailed flow of our Rank-DSE: (a) General exploration flow; (b) Detailed flow with Model construction and Sampling part emphasized.

In this work, the structure of an out-of-order microprocessor is modeled as a DAG in a similar way to capture microarchitectural interdependencies. As shown in the middle part of Figure 4(a), the superscalar microarchitecture is modeled as two subgraphs using a DAG: the front end and the execution core. The front end comprises two graph nodes: **instruction fetch** (**IF**) and **instruction decode** (**ID**), modeling IF, decoding, and dispatch. The Execution Core consists of five graph nodes: **physical register file** (**PRF**), INT (integer-related execution units), FP (floating point-related execution units), MEM (memory-related execution units), and LSU, which coordinates the execution of translated micro-operations and result writeback.

In the microarchitecture DAG, nodes represent components with key parameters embedded as node features $\boldsymbol{H}_x$, indicating component sizes with zeros padded to the same dimension as in Figure 4(b). On the other hand, each edge of the adjacency matrix $\boldsymbol{A}_x$ represents a specific dependency between microarchitecture components based on the processor's instruction flow, as in Figure 4(a). In this work, the micro-architecture is modeled as a DAG where nodes represent components with key parameters embedded as node features $\boldsymbol{H}_x$. For example, the IF node has features $\boldsymbol{H}_{x,\mathrm{IF}} = [4, 16, 32, \cdots, 0]$ representing 4-way fetch, 16-entry buffer, and 32-entry fetch queue. Edges in the adjacency matrix $\boldsymbol{A}_x$ capture dependencies between components based on instruction flow. Node attributes directly represent component-specific parameters, while edge

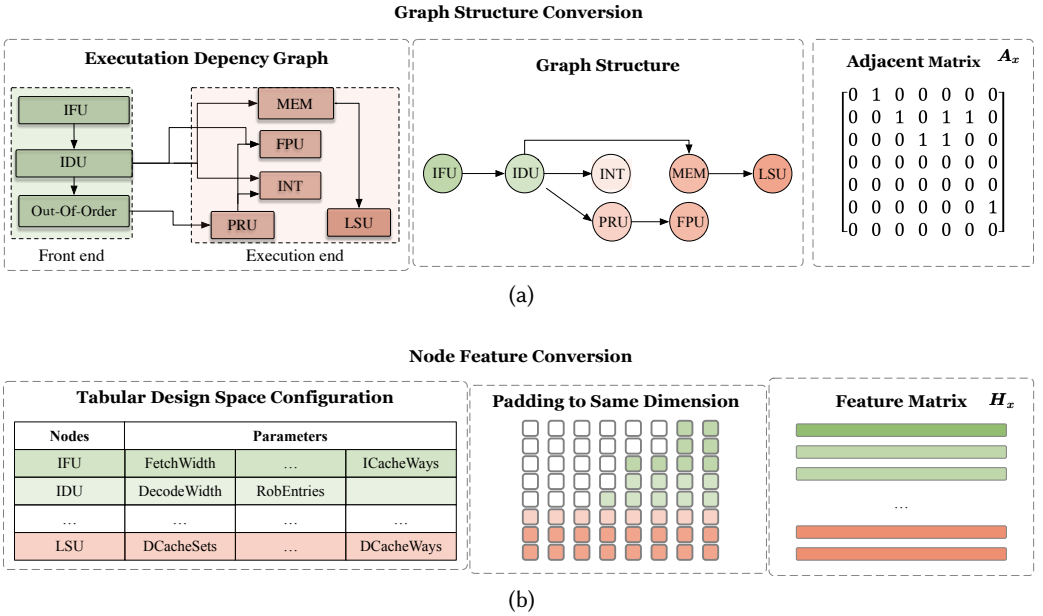


Fig. 4. Each architecture is represented as a DAG: (a) The edges represent the data dependency between two different nodes; (b) The nodes of the architecture DAG indicate the different units with key parameters, with zeros padded.

connections encode the data dependencies between components, as shown in the left figure of Figure 4(a). The directed edges also encode parameter inter-dependencies. The edges from ID to INT, FP, and MEM execution units capture the inter-dependencies between the execution units and the decode unit.

### 3.3 NPC

Previous methods have focused on predicting the absolute value of PPA metrics for different architecture designs. They use an **expected hypervolume improvement** (**EHVI**) metric, expected improvement to guide the exploration process [35]. However, the accuracy of PPA estimation is affected by limited training samples, leading to potential degradation of the exploration process. In our approach, instead of predicting the absolute value of PPA, we propose using relative predictions, which provide a more robust guide for exploration under limited data.

While previous methods have used L2R techniques to predict ranking results, they are limited to single-value cases and do not consider multi-objective optimization problems [12]. Concerning the multi-objective optimization problem, multiple metrics would be considered simultaneously. Typically, the dominant relationship between them would be employed to judge the priority of the two candidates. In our proposed NPC model, we capture the pairwise relations between architecture candidates. Instead of directly predicting rankings, NPC compares two architectures and predicts the probability of one dominating the other.

Specifically, given two architectures $\boldsymbol{x}$ and $\boldsymbol{x}'$ as inputs, the proposed NPC predicts the probability that $\boldsymbol{x}$ dominates $\boldsymbol{x}'$ as shown in Figure 5(a). NPC can compare any two architectures and predict the comparison probability as the expected improvement. The **graph convolution network** (**GCN**) encoder part of NPC exploits the input feature and connectivity information within the constructed

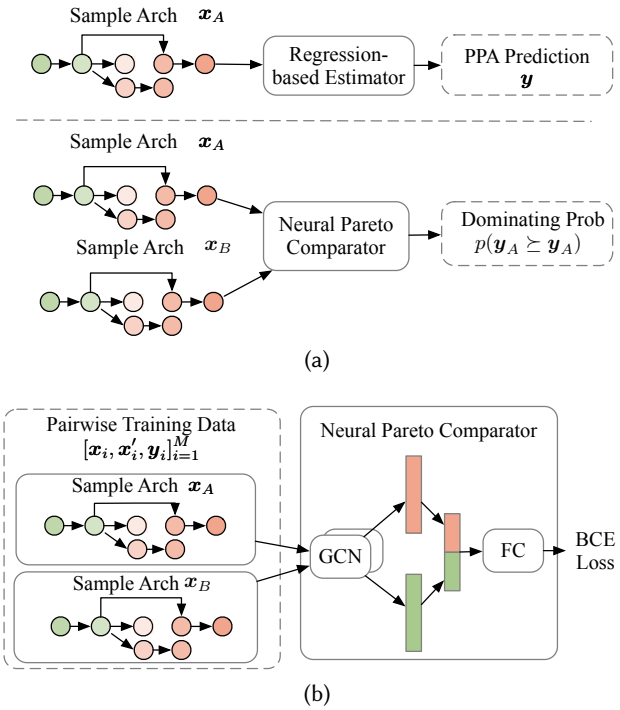


Fig. 5. The structure of our NPC. (a) The inference process of our NPC model: the NPC takes two architecture configurations as input and outputs the dominant probability, different from the regression-based model. (b) The training process of our NPC model: we collect the pairwise data of the dominant relationship between different architecture configurations, and use a BCE loss to train the NPC model.

graph. Based on the adjacent matrix and node attributes data pair $(\mathbf{A}_{\boldsymbol{x}}, \boldsymbol{H}_{\boldsymbol{x}})$, we use a two-layer GCN [36] to extract the architecture features $\boldsymbol{Z}_{\boldsymbol{x}}$:

$$\boldsymbol{Z}_{\boldsymbol{x}} = f(\boldsymbol{H}_{\boldsymbol{x}}, \mathbf{A}_{\boldsymbol{x}}) = \mathbf{A}_{\boldsymbol{x}} \phi\left(\mathbf{A}_{\boldsymbol{x}} \boldsymbol{H}_{\boldsymbol{x}} \boldsymbol{W}^{(0)}\right) \boldsymbol{W}^{(1)}, \tag{1}$$

where $\boldsymbol{W}^{(0)}$ and $\boldsymbol{W}^{(1)}$ denote the weights of graph convolutional layers, $\boldsymbol{H}_{\boldsymbol{x}}$ is the transformed feature of the original feature $\boldsymbol{x}$ using an embedding matrix, $\phi$ is the non-linear activation function, and $\boldsymbol{Z}_{\boldsymbol{x}}$ refers to the extracted features.

After extracting the two input architectures' information, the output embeddings of $\boldsymbol{x}$ and $\boldsymbol{x'}$ are sent to a **fully connected** (**FC**) layer as shown in Figure 5(b). The sigmoid function $\sigma(\cdot)$ takes the output of the FC layer as input and outputs the dominating probability between two architectures:

$$p = \mathbf{NPC}\left(\boldsymbol{x}, \boldsymbol{x'}\right) = \sigma\left(\left[\boldsymbol{Z}_{\boldsymbol{x}}; \boldsymbol{Z}_{\boldsymbol{x'}}\right] \boldsymbol{W}^{FC}\right), \tag{2}$$

where $Z_{\boldsymbol{x}}$ denotes the features extract from the architecture $\boldsymbol{x}$, $\boldsymbol{W}^{FC}$ denotes the weight of the FC layer, $[\boldsymbol{Z}_{\boldsymbol{x}}; \boldsymbol{Z}_{\boldsymbol{x'}}]$ refers to the concatenation of the features of $\boldsymbol{x}$ and $\boldsymbol{x'}$.

To train the proposed NPC, we are required to construct the comparison results dataset of architectures based on their corresponding PPA metrics. The label $y_i$, for each pair of designs $(x, x')$, is defined as $y_i = \boldsymbol{1}\{\boldsymbol{x} \succeq \boldsymbol{x'}\}$, and we construct the training data as $(\boldsymbol{xi}, \boldsymbol{xi'}, y_i)$. Consequently, training the NPC can be viewed as a binary classification problem, with the label $y$ belonging to the set $0, 1$. We solve this problem by optimizing the **binary cross-entropy** (**BCE**) loss, which

**ALGORITHM 1:** Semi-Supervised Learning for NPC

**Require:** Labeled data set $\mathcal{D}_L = \{(\boldsymbol{x}_i, \boldsymbol{x}_i', y_i)\}_{i=1}^{N_L}$, Unlabeled data set $\mathcal{D}_U = \{(\boldsymbol{x}_j, \boldsymbol{x}_j')\}_{j=1}^{N_U}$; learned architecture comparator NPC$(\cdot,\cdot)$, and parameter $K$.

```
Initialize D = ∅ and the confidence score set F = ∅;
for j = 1 to N_U do
    p_j = NPC(x_j, x'_j);
    y'_j = 1{p_j ≥ 0.5};                    ▷ Compute the predicted label
    Let D = D ∪ {x_j, x'_j, y'_j};
    f_j = |p_j − 0.5|;                      ▷ Compute confidence score
    Let F = F ∪ {f_j};
end for
D_semi = top-K(D, F);                       ▷ Select top-K pairs
D_L = D_L ∪ D_semi;
Train NPC on D_L;
```

measures the difference between the predicted probability $p$ by the NPC and the ground truth label $y$:

$$\text{BCE} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1-y_i)\log(1-p_i)]. \tag{3}$$

By comparing architectures instead of estimating the absolute performance, our proposed NPC model provides a more robust approach for guiding the exploration process in architecture design. Another advantage of the NPC model is that it has a lower requirement for the number of training samples. Given $m$ architectures with the corresponding performance, $\binom{m}{2} = \frac{m\times(m-1)}{2}$ training pairs can be used to train NPC, while the regression-based methods only have $m$ training samples.

### 3.4 Semi-supervised Learning Scheme

Traditional fully supervised learning approaches may struggle when extrapolating outside the labeled data distribution, limiting the exploration of unknown solutions. Those DSE methods involve exploring a vast design space to find optimal or near-optimal solutions. However, obtaining labeled data for training machine learning models in DSE can be expensive and time-consuming in many practical scenarios. In previous works, semi-supervised methods, including active learning [10], data engineering[11], and transfer learning [37], are employed to further improve the performance of predictors on untrained data.

In Rank-DSE, we develop a semi-supervised learning scheme to exploit unlabeled architecture samples to improve the generalized performance, as shown in Figure 6(a) and Algorithm 1. Specifically, for these unlabeled data, we propose to take the classes with maximum probabilities predicted by NPC as the pseudo labels. As shown in Algorithm 1, given the latest NPC model NPC$(\cdot,\cdot)$ for each epoch, the predicted label of the previously unseen architecture pairs can be computed by

$$y' = 1\{\text{NPC}(\boldsymbol{x}, \boldsymbol{x}') \geq p\}, \tag{4}$$

where 1 refers to an indicator function. Here, $y' = 1$ if NPC$(\boldsymbol{x}, \boldsymbol{x}') \geq p$ and $y' = 0$, otherwise.

To mitigate the issue of potentially inaccurate predictions from the NPC model, we employ a strategy to assess the reliability of the predicted label for each architecture pair. This evaluation involves calculating a confidence score, which serves as a measure of the prediction's quality. For the $k$th architecture pair, we compute the confidence score as

$$f_k = \left|\text{NPC}\left(\boldsymbol{x}_k, \boldsymbol{x}_k'\right) - p\right|. \tag{5}$$

**ALGORITHM 2:** Update Baseline Pareto Front for Rank-DSE

**Require:** Existing baseline architectures $\mathcal{D}_B$, sampled architectures $\mathcal{D}_S$, and learned architecture comparator NPC$(\cdot,\cdot)$, the size of baseline set $N_B$.

1: $\mathcal{H} = \emptyset$; ▷ Initialize comparison score
2: Construct a candidate baseline set $\mathcal{D}_B = \mathcal{D}_B \cup \mathcal{D}_S = \{\alpha_i\}_{i=1}^{|\mathcal{D}_H|}$;
3: **for** $j = 1$ to $|\mathcal{D}_B|$ **do**
4: $r_i = \frac{1}{|\mathcal{D}_B|-1} \sum_{1\le j\le|\mathcal{D}_H|, i\neq j} \text{NPC}\left(\alpha_i, \alpha_j\right)$; ▷ Compute average score
5: Let $\mathcal{H} = \mathcal{H} \ \{s_i\}$;
6: **end for**
7: $\mathcal{D}_B = \text{top-}N_B(\mathcal{D}_B, \mathcal{H})$; ▷ Select top-$N_B$ samples as baseline set

**Ensure:** $\mathcal{D}_B$.

In practical applications, a higher confidence score indicates a higher level of reliability in the predicted label. We adopt a selection process where we choose the data with the top-$K$ confidence scores as the predicted labels and combine them with the labeled data to train the NPC model. To maintain a balance between these two types of data, we set the maximum proportion of semi-supervised data to $p = 0.5$. As the amount of unlabeled data increases during training, our NPC model becomes better equipped to generalize to unseen architectures and improve its overall performance.

Early training stages are vulnerable to model instability and unreliable pseudo-labels. Rank-DSE addresses this through: (1) Iterative Refinement. Pseudo-labels are updated in each epoch using the latest NPC model. The model's predictive accuracy improves as training progresses, reducing reliance on early-stage errors; (2) Confidence-Guided Data Augmentation. Prioritizing high-confidence pairs ensures that initial pseudo-labels are more reliable; (3) Balanced Data Mixing. The top-$k$ semi-supervised cap ensures labeled data anchors the model during early training, preventing drift from noisy pseudo-labels.

### 3.5 RL Aided Sampling Strategy

In Sections 3.3 and 3.4, we introduce a comparison mapping function, i.e., NPC. This mapping allows us to compare any two architectures, denoted as $\boldsymbol{x}$ and $\boldsymbol{x'}$, belonging to the design space $\mathcal{D}$. NPC outputs the probability of $\boldsymbol{x'}$ dominating $\boldsymbol{x}$, which we denote as $p$:

$$p = Pr[\boldsymbol{x} \succeq \boldsymbol{x'}] = \text{NPC}\left(\boldsymbol{x}, \boldsymbol{x'}\right). \tag{6}$$

From Equation (6), we can utilize the predicted comparison probability from NPC as the expected improvement to search for the Pareto front within the given design space $\mathcal{D}$. Achieving this typically involves utilizing the NPC as a comparator to perform non-dominated sorting across the entire design space. However, due to the necessity of comparing each configuration with every other, the complexity of this deduction process often scales as $O(N^2)$ in terms of the number of sampled designs $N$. Given that the real design space is frequently exponentially large, conducting direct comparisons across the entire design space becomes challenging. Consequently, employing an effective sampling scheme becomes crucial in reducing required comparisons to attain high-quality Pareto solutions at a limited cost.

To solve the problem described above, we train a controller to sample potential Pareto solutions with a policy gradient [38] with updating baseline [39], as shown in Figure 6(b) and Algorithm 2. Formally, given a baseline architecture design $\boldsymbol{x}_\beta$, we learn a policy $\pi_\theta(\boldsymbol{x})$ by solving the following optimization problem:

$$\max_\theta \mathbb{E}_{\alpha\sim\pi_\theta(\alpha)} \Pr\left[\boldsymbol{x}_\alpha \succeq \boldsymbol{x}_\beta\right], \tag{7}$$

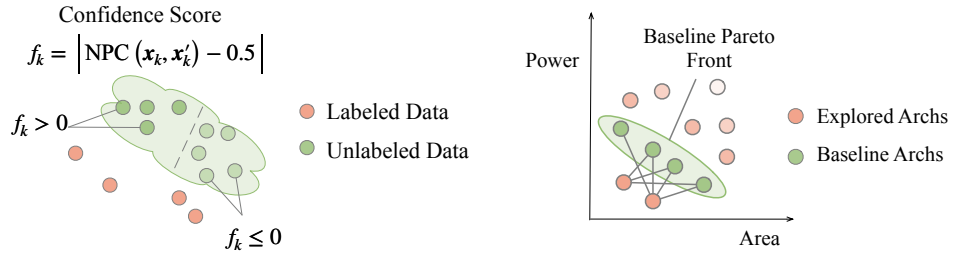


Fig. 6. (a) Semi-supervised Learning: It uses confidence scores to select unlabeled architectures, assigning pseudo labels based on the highest confidence class. (b) RL Controller: This controller samples potential Pareto solutions using policy gradients, with rewards based on the average comparison probability with the Baseline Pareto front.

where $\theta$ denotes the parameters of the policy. Given a specific baseline architecture $\boldsymbol{x}_\beta$, the controller seeks to conduct a comparison with it to search for better architectures.

To maximize the expected reward in Equation (7), we use the Monte Carlo estimate to compute the gradient as in [40]. We first sample $N$ architectures $\{x_j\}_{j=1}^N$ by sampling from the parameterized distribution $x \sim \pi_\theta(\cdot)$. Then we update the parameter $\theta$ using policy gradient:

$$\theta \leftarrow \theta + \eta \frac{1}{N} \sum_{j=1}^{N} \left[\nabla_\theta \log \pi_\theta(x_j) \text{NPC}(x_j, x_\beta)\right], \tag{8}$$

where $N$ is the number of different architecture pairs that the controller samples in one batch, and $\eta$ is the learning rate for the policy network.

To improve the baseline architecture $\boldsymbol{x}_\beta$, our objective is to identify the most optimal architecture from a pool of previously searched architectures. This involves constructing a candidate baseline set $\mathcal{D}_B$ and dynamically incorporating sampled architectures into it. To mitigate the potential negative impact resulting from a single comparison error, we adopt an averaging approach. For each architecture $\boldsymbol{x}_i$ within $\mathcal{D}_B$, we calculate the average comparison probability $r_i$ by comparing $\boldsymbol{x}_i$ with the other architectures in $\mathcal{D}_B$ using the NPC model:

$$r_i = \frac{1}{|\mathcal{D}_B| - 1} \sum_{1 \le j \le |\mathcal{D}_{\mathcal{H}}|} \text{NPC}\left(\boldsymbol{x}_i, \boldsymbol{x}_j\right). \tag{9}$$

Using the best architecture observed thus far as the baseline Pareto front set, our Rank-DSE methodology significantly improves the search performance by identifying architectures that surpass the baseline architectures in terms of PPA metrics.

### 3.6 Overall Tuning DSE Framework

The detailed algorithm of our RankDSE framework is described in Algorithm 3. The RankDSE framework proceeds by sequentially invoking the Optimize iteration (lines 6–13). The Optimize algorithm first trains the NPC model with the labeled data $\mathcal{D}_L$ (line 6). Then, it iteratively performs RL aided sampling (lines 7–8), semi-supervised learning scheme (lines 9–10), and update (lines 11–13). The iterative optimization steps will stop when reaching the maximum iterations.

### 3.7 The Complexity Comparison between Different Methods

DSE methods vary significantly in complexity and practicality [10, 11, 41–43].

Table 2. The Comparison and Analysis between Different DSE Methods

| Method | Time | Space | Scalability | Use Case |
|---|---|---|---|---|
| Exhaustive Search [41] | $O(N_c^d)$ | $O(1)$ | Limited | Small spaces. |
| Genetic Algorithm [44] | $O(MN_p^2)$ | $O(MN_p)$ | Medium | Medium spaces. |
| Regression-based Methods [30, 31] | $O(p^2n)$–$O(n^3)$ | $O(p^2)$–$O(n^2)$ | Moderate-dimensional | Interpretable predictions. |
| Bayesian Optimization [10, 11] | $O(n_t^3)$ | $O(n_t^2)$ | High-dimensional spaces | Statistical evaluations. |
| RL [45] | $O(T \cdot A \cdot S)$ | $O(S \cdot A)$ | Dynamic tasks | Adaptive decision-making. |

**ALGORITHM 3:** The Overall Algorithm of RankDSE

**Require:** Learning rate $\eta$ for policy gradient, parameters $N_U$ for number of explored configs and $N_B$ for size of baseline Pareto front.

**Ensure:** Pareto-optimal architecture configs set $\mathcal{P}$

1: Construct training data $\mathcal{D}_0 = \left\{\left(\boldsymbol{x}_i, \boldsymbol{x}_i', y_i\right)\right\}_{i=1}^{M(M-1)/2}$ for NPC by traversing all pairwise combinations;
2: Initialize parameters $\theta$ for $\pi(;;\theta)$ and $\boldsymbol{w}$ for NPC;
3: Initialize the baseline architecture $\beta \sim \pi(\cdot;\theta)$;
4: Let $\mathcal{D}_L = \mathcal{D}_0, \mathcal{D}_B = \text{Pareto\_Front}(\mathcal{D}_0), \mathcal{P} = \emptyset$;
5: **for** $t = 1, \ldots, T$ do **do**
6: Train NPC with data $\mathcal{D}_L = \left\{\left(\boldsymbol{x}_i, \boldsymbol{x}_1', y_i\right)\right\}_{i=1}^{|\mathcal{D}_L|}$;
7: Sample $N_B$ configs $\left\{\boldsymbol{x}_j\right\}_{j=1}^{N}$ by $\boldsymbol{x} \sim \pi_\theta(\cdot)$ as $\mathcal{D}_S$;
8: $\theta \leftarrow \theta + \eta \frac{1}{N} \sum_{j=1}^{N} \left[\nabla_\theta \log \pi\left(\boldsymbol{x}_j\right) \text{NPC}\left(\boldsymbol{x}_j, \boldsymbol{x}_\beta\right)\right]$;
9: Sample $N_U$ architectures $\mathcal{D}_U = \{\boldsymbol{x}_i\}_{i=1}^{N} \sim \pi_\theta(\cdot)$;
10: Construct new $\mathcal{D}_{semi}$ with $\mathcal{D}_U$ using Algorithm 1;
11: Update the baseline $\mathcal{D}_B$ using Algorithm 2;
12: Update the labelled dataset $\mathcal{D}_L = \mathcal{D}_L \cup \mathcal{D}_{semi}$;
13: Update the accumulated Patero Front $\mathcal{P}$ with $\mathcal{D}_S$;
14: **end for**

We analyze and compare the complexity of different DSE methods as in Table 2. Exhaustive Search, like random search [41], guarantees global optimality but suffers from exponential time complexity ($O(N_c^d)$) with $N_c$ the number of config candidates and $d$ the dimension size. The low efficiency when extending to high-dimensional situations makes it only applicable for small-scale problems ($d < 5$). Genetic Algorithms (e.g., NSGA-II [44]) scale quadratically ($O(MN_p^2)$) with population size, where $M$ is the number of objectives and $N_p$ is the population size, balancing runtime and solution quality for medium-scale multi-objective optimization. Bayesian Optimization [10, 11], with cubic time complexity ($O(n_t^3)$) and quadratic space overhead ($O(n_t^2)$) with $n_t$ as the number of sample points, excels in high-dimensional, expensive evaluation scenarios by leveraging surrogate models. RL [45] adapts to dynamic, sequential tasks but incurs linear time/space costs tied to state-action spaces ($O(T \cdot A \cdot S)$), often requiring longer training times, with $T$ the time steps and $S \cdot A$ is the size of the state-action space. Regression-based selection methods, such as LR [46] or SVR [30], use parametric or kernel-based models to predict design performance, offering time complexity between $O(p^2n + p^3)$ (LR, for $p$ features and $n$ samples) and $O(n^3)$ (SVR with nonlinear kernels).

These methods trade interpretability (linear models) or flexibility (SVR) for reduced computational overhead, but may struggle with highly nonlinear or high-dimensional spaces compared to Bayesian Optimization. While Exhaustive Search and Bayesian Optimization represent extremes (exact but

impractical vs. approximate but efficient), Genetic algorithms, regression-based selection, and RL offer tradeoffs between scalability, adaptability, and interpretability, with choice depending on problem size, evaluation cost, and decision-making needs. Our methods can be classified into the RL-based DSE methods with a customized ranking-based reward prediction model.

## 4 Experimental Results

This section evaluates the proposed NPC by comparing it with the state-of-the-art processor microarchitecture DSE algorithm and the ICCAD Contest 2022 winners.

### 4.1 Experimental Setting

**Design Tasks**. The experiments are performed on the design space $\mathcal{D}$ as in the ICCAD Contest 2022, which consists of 15, 633 different microarchitectures and corresponding PPA metric values [1]. To ensure versatility across diverse applications, the performance and power values are computed as averages based on multiple benchmarks, rather than relying on a single test case. The runtime cost of the VLSI flow includes RTL generation, logic synthesis, netlist simulation, power analysis, and so on. For different scales of microarchitectures, the runtime cost varies.

**Compared Methods**. The baseline methods to be compared include the following:

- Manual Designs: SonicBOOM, developed by senior engineers [47] serves as a baseline for comparison, representing the traditional manual design approach by senior engineers. It reflects the expertise and experience of human designers in crafting high-performance microarchitectures.
- Random Search [41] is included as a simple yet effective exploration method. It helps to establish a baseline for the efficiency and effectiveness of more sophisticated DSE algorithms by providing a basic level of exploration without any inherent bias or guidance.
- LR [46] and GP [48] are representative of model-based approaches. They are the typical works to utilize regression-based models to predict and optimize microarchitecture performance.
- NSGA-II: A multi-objective genetic algorithm that explores the design space with trade-off between critical metrics, thus, producing a set of Pareto-optimal solutions for diverse application requirements [44].
- ICCAD'21 (Boom-Explorer): A method based on BOOM architecture for exploring the microarchitecture design space, which won ICCAD 2021 Best Paper [10].
- GRL-DSE represents a more recent and innovative approach that incorporates prior knowledge through a graph autoencoder. A guided DSE uses graph autoencoder to embed the prior knowledge [11].
- ICCAD CAD Contest 2022 winners are included to represent the cutting-edge solutions developed in a competitive setting. Their inclusion helps to validate the competitiveness and practical applicability.

These methods demonstrated outstanding performance in the competition and are considered strong baseline approaches.

**Evaluation Metrics**. In experiments, we evaluate all methods based on four metrics, Pareto HV, exploration cost (Cost), and score as in the ICCAD CAD Contest 2022 [1], and **Kendall's Tau** (**KTau**) for evaluating the ranking accuracy for different models [49]. Pareto HV as illustrated in Equation (10), is the Lebesgue measure of the space dominated by the Pareto frontier and bounded by a reference point $\boldsymbol{v}_{ref}$.

$$\mathrm{HV}_{v_{\mathrm{ref}}}\left(\mathcal{P}\left(\mathcal{Y}\right)\right)=\int_{\mathcal{Y}}\mathbb{1}\left[y\succcurlyeq v_{\mathrm{ref}}\right]\left[1-\prod_{y_{*}\in\mathcal{P}\left(\mathcal{Y}\right)}\mathbb{1}[y_{*}\not\succeq y]\right]\mathrm{d}y, \tag{10}$$

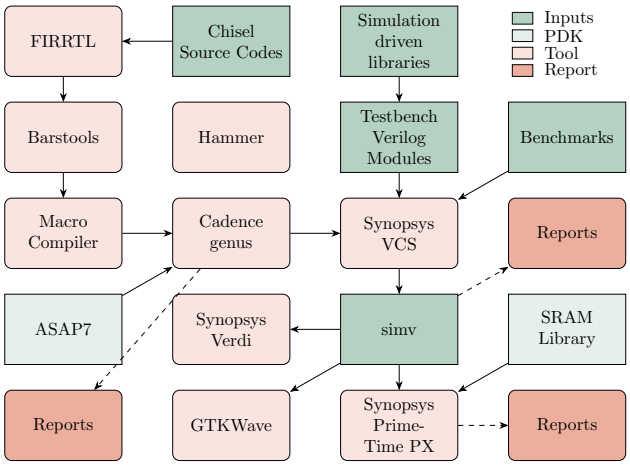


Fig. 7. The illustration example of the VLSI evaluation flow.

where $\mathbb{1}(\cdot)$ is the indicator function, which outputs 1 if its argument is true and 0 otherwise, $\mathcal{P}(\mathcal{Y})$ is the Pareto frontier. Exploration cost (Cost) measures the total time of algorithms, including the submission of contestants' algorithms and the time spent on the VLSI flow. Total score (Score) is the approximate scoring function of the ICCAD contest designs based on industry experience, w.r.t. Pareto HV and ORT:

$$\text{score} = \text{HV}_{v_{\text{ref}}} \cdot \begin{cases} \alpha - \frac{\text{ORT}-\theta}{\theta}, & \text{ORT} \geq \theta; \\ \alpha + \left|\frac{\text{ORT}-\theta}{\theta}\right|, & \text{ORT} < \theta, \end{cases} \tag{11}$$

where $\alpha$ is an exploration cost baseline, equal to 6, and $\theta$ is a pre-defined Cost budget, equivalent to 2,625,000. KTau is suitable for comparing the ranking accuracy of different predictors, where a higher value indicates better ranking results. KTau is defined as

$$\text{KTau} = 2 \times \frac{\text{number of concordant pairs}}{C_n^2} - 1, \tag{12}$$

where $n$ is the number of samples, and $C_n^2 = \frac{n(n-1)}{2}$. A concordant pair means that the rankings of predicted values and the actual values of a given pair are the same. The typical range of KTau is between -1 and 1.

**Training Configuration**. In our experiments, we adopt a 2-layer GCN with *hidden_size* $= 64$ as the backbone of NPC, and *ReLU* as the activation function. We adopt an LSTM network with *hidden_size* $= 64$ as the policy network. At the training stage, we adopted Adam as the default optimizer with a learning rate of 0.005 and 0.0002 for NPC and the policy network, respectively. The training epochs of NPC are fixed at 100, with cosine decay to adjust the learning rate.

**The VLSI Evaluation Flow**. As shown in Figure 7, we adopted the evaluation results from the ICCAD Contest 2022 platform [1]. For each configuration, the VLSI Flow is adopted to explore and optimize the performance of the BOOM [47] through a structured, simulation-driven process. It begins with Chisel [51] source code compiled to FIRRTL [52, 53] and then to synthesizable Verilog using Barstools [54]. Cadence Genus performs logic synthesis with the ASAP7 PDK and SRAM Library [55], generating reports on critical-path delay and power estimates. Synopsys VCS runs functional simulations with testbench modules and benchmarks, producing reports and waveforms

Table 3. Performance Comparison Among Different Methods

| Methodologies | Methods | HV | Cost | Score |
|---|---|---|---|---|
| Sample-based | Random Search [41] | 6.82 | 469,658.03 | 46.49 |
| Genetic Algorithm | NSGA-II [44] | 6.84 | 1,028,259.39 | 45.20 |
| Regression-based | LR [46] | 7.65 | 1,630,733.98 | 48.78 |
| | GP [48] | 7.66 | 2,389,863.42 | 46.65 |
| | MO-TPE [50] | 6.98 | 1,254,281.21 | 45.49 |
| | ICCAD'21 [10] | 7.31 | 564,941.34 | 49.59 |
| Ranking-based | **Ours** | **7.82** | **299,914.65** | **53.85** |

Table 4. Score Comparison between GRL-DSE and ICCAD Contest Winners

| Winner | Score |
|---|---|
| Top1 | 52.09 |
| Top2 | 51.40 |
| Top3 | 51.13 |
| GRL-DSE [11] | 52.22 |
| **Ours** | **53.85** |

for analysis. Synopsys PrimeTime PX conducts static timing analysis and power estimation. The flow leverages tools to quantify performance metrics like timing, power, area, and throughput. Iterative refinement based on reports and simulation results enables fixes for timing violations, power reduction, area optimization, and improved throughput, with benchmarks stress-testing the design under realistic conditions. This systematic evaluation of performance metrics throughout the flow facilitates efficient DSE and ensures BOOM achieves its performance targets.

### 4.2 Experimental Results

The normalized results of HV, Cost, and score can be found in Tables 3 and 4. In terms of the Score metric, Rank-DSE outperforms LR, GP, MO-TPE [50], and ICCAD'21 by 10%, 15%, 18%, and 8%, respectively. Specifically, our Rank-DSE method achieves superior HV values while utilizing fewer costs, surpassing LR, GP, MO-TPE [50], and ICCAD'21 by 2%, 2%, 12%, and 7%, respectively, in the HV metric. Moreover, Rank-DSE requires only 18.39%, 12.55%, 23.91%, and 53.09% of the exploration cost compared to LR, GP, MO-TPE [50], and ICCAD'21, respectively. These findings indicate that our ranking-based approach achieves more robust prediction guidance with a smaller sample cost, leading to enhancement in solution quality.

According to Table 4, our method achieved the highest score of 53.85, surpassing the current top-3 winners and the GRL-DSE method [11] in the ICCAD CAD Contest 2022. Compared to the top-1 solution, our score is 3.38% higher, indicating a significant difference. Additionally, compared to the GRL-DSE method, our score is higher by 3.12%. Most methods, including GRL-DSE, use regression-based predictors to guide the search process. By surpassing the winners, our method validates the effectiveness of the ranking-based approach. It demonstrates better robustness and greater efficiency, thereby outperforming other competitors. These characteristics provide significant advantages to our method in practical applications.

We also visualize the Pareto-optimal sets obtained from the baselines and Rank-DSE of Table 3 in the ICCAD CAD Contest 2022, as shown in Figure 8. Each point represents a solution where both objectives are minimized. The curve in Figure 8(a) shows the tradeoff between performance and area, with solutions along the curve representing the best possible balance where improving one objective would lead to a degradation in the other. Similar to subfigure Figure 8(a), the curve in Figure 8(b) indicates the optimal balance between performance and power consumption, and Figure 8(c) represents the optimal tradeoff between power and area. We utilized the ICCAD Contest data [1], where the contest platform has normalized all PPA values for microarchitectures. According to the contest rules, *a better microarchitecture boasts higher performance but has lower power dissipation and a smaller area.* Consequently, when normalizing the performance metric, they adjusted the sign corresponding to the regularization. For the performance metric, a larger normalized relative score indicates fewer clock cycles. Nonetheless, normalized area ($\mu m^2$) and normalized power ($nW$) still correspond to minimization cases. The results indicate that Rank-DSE exhibits a significant

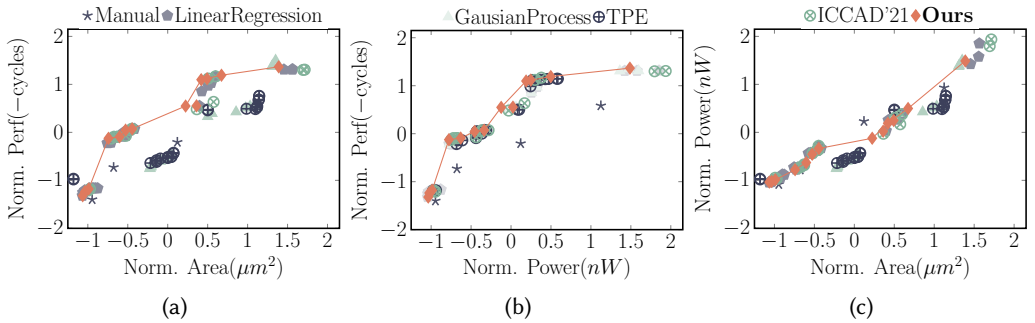


Fig. 8. The visualization of the searched Pareto front of different methods: (a) The learned Pareto optimal set (normalized performance *vs.* normalized chip area); (b) The learned Pareto optimal set (normalized performance *vs.* normalized power consumption); (c) The learned Pareto optimal set (normalized power consumption *vs.* normalized chip area).

dominance over the Pareto front learned by all other baseline methods, resulting in a notable performance improvement compared to the baselines.

The Rank-DSE framework offers a notable advantage in reducing exploration costs. This advantage can be verified by referring to Figure 9. The figure showcases the obtained HV at different exploration costs. In particular, Rank-DSE outperforms the other methods by producing a solution set with better HV at the same cost. For instance, when the cost is around $3 \times 10^4$, Rank-DSE achieves an HV performance that surpasses the other methods. Additionally, when comparing the cost required to achieve the same HV, Rank-DSE demonstrates remarkable efficiency compared to other methods. Specifically, Rank-DSE only needs to utilize around 20% of the exploration cost to achieve an HV similar to that of ICCAD'21.

### 4.3 Comparison with Manual BOOM Microarchitecture Design

We selected our Pareto design from the Rank-DSE Pareto-optimal set. It is then compared to a SonicBOOM developed by senior engineers [47]. To demonstrate the superiority of our Pareto design over the Mega SonicBOOM, we evaluated both designs using multiple benchmarks. These benchmarks assess the efficiency of integer arithmetic, vector arithmetic, sorting, matrix multiplication, recursive problem solving, and sparse matrix-vector operation, respectively.

These provide a standardized way to measure and compare the performance of different computing systems and algorithms. These benchmarks encompass a range of applications, including basic instructions from the ISA (Instruction Set Architecture), such as multiply and add, and DSP-oriented algorithms like air and firm are considered. Furthermore, real-time computing applications, such as qsort and resort, and duff, are also included.

Dhrystone, Median, MM, MT-Matmul, MT-VVADD, Multiply, Qsort, Rsort, SPMV, Towers, and VVADD are different benchmark tests used to evaluate the performance of various algorithms and operations in computer systems. They assess the efficiency of integer arithmetic, sorting, matrix multiplication, vector addition, integer multiplication, quicksort, recursive sorting, sparse matrix-vector multiplication, Tower of Hanoi problem solving, and vector addition algorithms, respectively. These benchmarks provide a standardized way to measure and compare the performance of different computing systems and algorithms.

Table 5 presents the average power consumption and clock cycles for all the selected benchmarks, which cover various application scenarios as in [10]. Figure 10 shows the comparison of power and performance between them. Across all the benchmarks, our Pareto design achieves approximately a

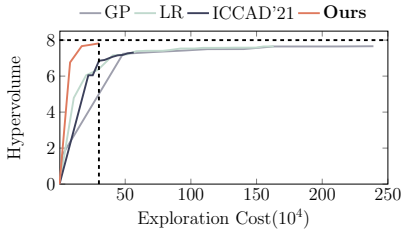


Fig. 9. The attained HV *vs.* exploration cost of different methods.

Table 5. Comparison with Mega SonicBOOM and the Searched Result of Rank-DSE

| Microarchitecture | Average Power (unit: watts) | Average Clock Cycles |
|---|---|---|
| Mega SonicBOOM | $6.0700 \times 10^{-2}$ | 50,529.9091 |
| Pareto Design * | $5.5892 \times 10^{-2}$ | 48,216.4546 |

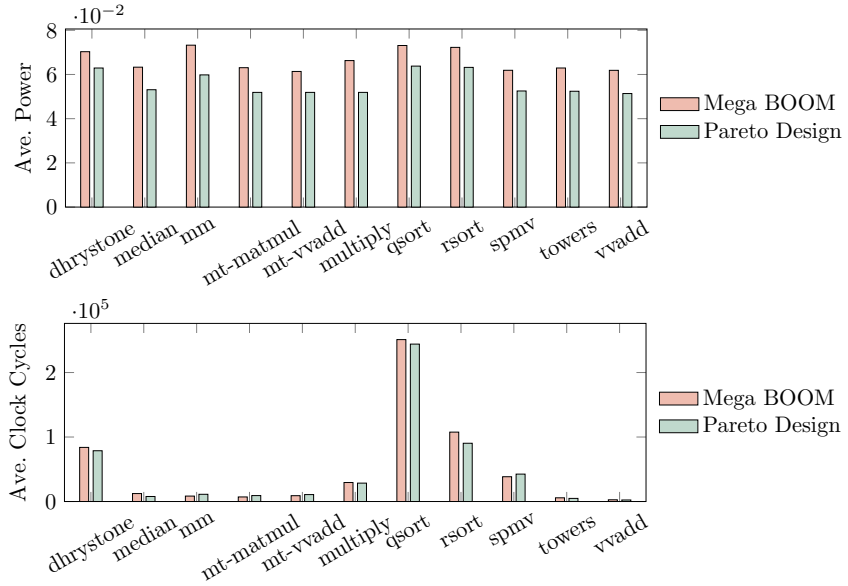


Fig. 10. Comparisons of power and performance between the pareto design found by rank-DSE and the mega SonicBOOM.

2.11% faster performance while reducing 3.45% less power consumption compared to the Mega SonicBOOM. While reducing redundant hardware resources while increasing necessary components, our Pareto design achieves a better tradeoff on power and performance. While Rank-DSE may not universally outperform Mega SonicBOOM in all benchmarks, as shown in Figure 10, it excels in scenarios requiring multi-objective tradeoffs and complex DSE, as shown in Table 5. Besides, the design time for Mega SonicBOOM is also more than the Rank-DSE searching process. Its automated approach is particularly beneficial for rapid prototyping and customization. However, further work is needed to integrate expert knowledge into the search process, enhance surrogate models for more accurate predictions. This integration of automated exploration with human expertise could bridge the performance gap and enhance Rank-DSE's effectiveness.

### 4.4 The Runtime Comparisons with VLSI Flow Evaluation Cost

As shown in Table 6 and Figure 11, the NPC training and RL policy consume minimal time compared to the VLSI Cost. We carried out a trial comparing the time taken by various components in the NPC with that of the VLSI Flow. Across all iterations, the combined time for NPC training and RL

Table 6. Runtime Comparison between Different Components of NPC with VLSI Flow Cost Across Different Iterations in Seconds ($s$)

| Iteration | Time ($s$) | | |
|---|---|---|---|
| | NPC Train | RL Policy | VLSI Cost |
| 1 | 0.0004 | 19.4 | 80,381.6 |
| 2 | 2.21 | 37.9 | 218,626.5 |
| 3 | 4.58 | 56.4 | 261,008.1 |
| 4 | 6.68 | 75.5 | 341,799.8 |
| 5 | 8.88 | 94.1 | 427,020.4 |
| 6 | 11.08 | 112.8 | 543,650.3 |
| 7 | 13.35 | 131.4 | 663,210.7 |
| 8 | 15.64 | 150.4 | 732,634.3 |
| 9 | 17.91 | 168.9 | 852,351.4 |
| 10 | 20.17 | 187.4 | 971,873.0 |

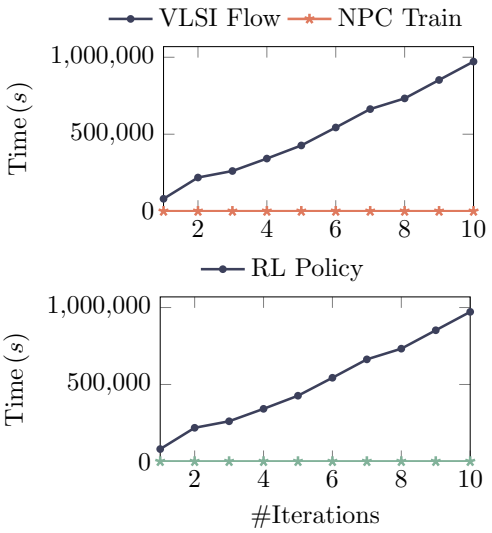


Fig. 11. Visualization of the runtime comparison between NPC training and RL policy (training and inference) with the VLSI flow evaluation cost.

remains a tiny fraction of the total time, with the VLSI Cost consistently dominating. For example, even at the 10-th iteration, the combined time for NPC training (20.17 seconds) and RL (187.4 seconds) is negligible compared to the VLSI Cost (971, 873.0 seconds). This pattern holds throughout all iterations, highlighting that NPC training and RL strategies do not significantly contribute to the overall time consumption. Besides, the solutions from the NPC and RL Policy achieved better quality, and thus the negligible overhead caused by them was well justified.

### 4.5 Ablation Studies

**The Effectiveness of Each Module of Rank-DSE**. To verify the effectiveness of the proposed modules in Rank-DSE, we conduct experiments by removing each one of them, as shown in Figure 12.

Here, ✘ GCN represents the Rank-DSE framework that uses a 2-layer MLP rather than the GCN (introduced in Section 3.3) as the encoder. ✘ Semi represents not consider the semi-supervised learning scheme (introduced in Section 3.4). ✘ RL represents replace the reinforcement learning sampler (introduced in Section 3.5) with a basic sampler selected architecture designs randomly.

We can observe that, without any of them, the obtained score drops by a substantial margin, demonstrating the effectiveness of all of them. Additionally, we can observe that the RL module contributes the most to the overall score, while the GCN and Semi-supervised modules contribute to a similar extent. This indicates the importance of an effective sampling method in ranking-based DSE methods.

**Predictor Performance Comparison between Different Methods**. One of the fundamental ideas in Rank-DSE is that the ranking of predicted values is more useful than their absolute values, especially when incorporating the predictors into the DSE process. In this section, we quantitatively compared the ranking prediction accuracy of different prediction models. Our proposed Ranking method is compared with methods based on LR and GPs. 1,000 data points were sampled from the ICCAD CAD 2022 dataset for training and testing.

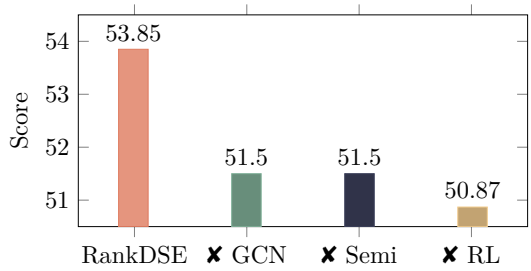


Fig. 12. The visualization of KTau of RankDSE and other regression-based estimators across different proportions of training samples for performance evaluation.

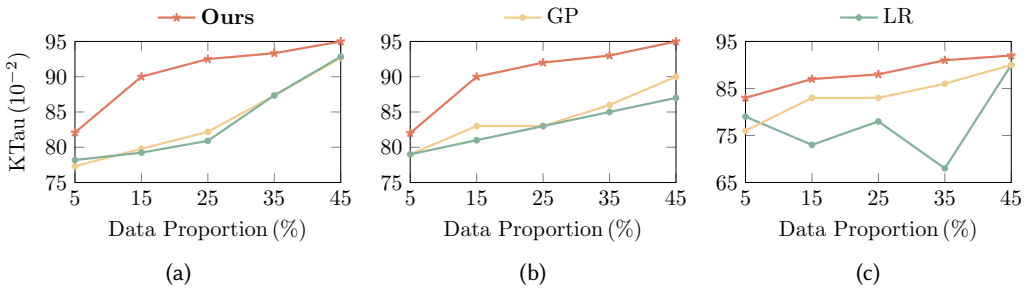


Fig. 13. The visualization of KTau of RankDSE and other regression-based estimators across different proportions of training samples: (a) Performance evaluation; (b) Power evaluation; (c) Area evaluation.

Table 7. The KTau of the Proposed Algorithms on the ICCAD Dataset (1,000 points) with Different Proportions of Training Samples

| Metric | Method | 5% | 15% | 25% | 35% | 45% |
|---|---|---|---|---|---|---|
| Perfermance | LR | 0.77 | 0.80 | 0.82 | 0.87 | 0.92 |
| | GP | 0.78 | 0.79 | 0.81 | 0.87 | 0.93 |
| | **Ours** | **0.82** | **0.90** | **0.92** | **0.93** | **0.95** |
| Power | LR | 0.79 | 0.81 | 0.83 | 0.85 | 0.87 |
| | GP | 0.79 | 0.83 | 0.83 | 0.86 | 0.90 |
| | **Ours** | **0.85** | **0.88** | **0.90** | **0.91** | **0.93** |
| Area | LR | 0.79 | 0.73 | 0.78 | 0.68 | 0.90 |
| | GP | 0.76 | 0.83 | 0.83 | 0.86 | 0.90 |
| | **Ours** | **0.83** | **0.87** | **0.88** | **0.91** | **0.92** |

Regarding evaluation metrics, we need to reflect the ranking accuracy of different predictors. Here, we chose KTau as the evaluation criterion. It is worth noting that the definition of KTau is designed for a single metric. Therefore, we establish separate prediction models for the three PPA metrics to measure the rank correlation of different predictors.

The experimental results are presented in Table 7 and Figure 13. In the DSE process, we typically use a small portion of the data, so we focus on the third column here with 5% of data points as the training dataset. We evaluate the ranking accuracy at different proportions for a comprehensive comparison. The final results demonstrate that our proposed NPC prediction model outperforms comparative methods like LR and the GP. It achieves an average improvement of over 6% in

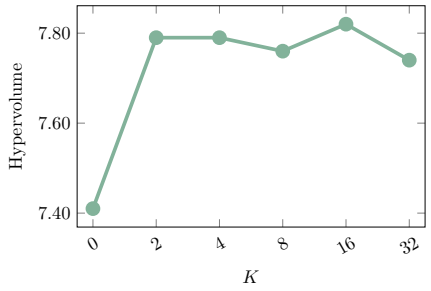


Fig. 14. The comparison of different hyperparameter $K$ of semi-supervised learning in our Rank-DSE framework.

| $p$ \ $K$ | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|
| 0.4 | 7.79 | 7.79 | 7.77 | 7.75 | 7.71 |
| 0.5 | 7.79 | 7.79 | 7.76 | 7.82 | 7.74 |
| 0.6 | 7.74 | 7.77 | 7.63 | 7.79 | 7.83 |
| 0.7 | 7.81 | 7.70 | 7.79 | 7.76 | 7.72 |
| 0.8 | 7.74 | 7.76 | 7.74 | 7.72 | 7.71 |

Fig. 15. The effect of the hyperparameter $K$ and threshold $p$ of semi-supervised learning in our rank-DSE framework.

predicting three metrics. Moreover, our prediction model consistently delivers higher accuracy across all three metrics. This indicates that our comparison-based ranking model, Rank-DSE, excels in predicting the ranking of architecture configurations.

**The Effect of the Hyperparameter in Semi-supervised**. To investigate the effect of our proposed semi-supervised learning method, we conducted more experiments on the design space of the ICCAD Contest 2022. As shown in Figure 14, compared with that without data exploration ($K = 0$), Rank-DSE achieves better HV results when $K$ is set to 16. Similarly, we also compared the HV results of different parameters from $K = 2$ to $K = 32$, as shown in Figure 15. A too low or too high $K$ would affect the training of the NPC model in Rank-DSE, so we chose $K = 16$. A threshold $p$ with a value that is too low or too high would also hinder the training of the NPC model in Rank-DSE, leading to poor search performance. Therefore, we set the threshold $p$ to 0.5 in the experiments.

## 5 Conclusion

In this article, we have proposed Rank-DSE, a novel framework that performs the exploration process based on comparison results of the sample architecture design and the baseline Pareto Front set. To provide the comparison results of multiple metrics, we design a **Neural Parato Comparator** (**NPC**) that takes two architectures as inputs and outputs the probability that one dominates the other. Moreover, we propose a semi-supervised way to utilize the power of unlabeled data. To further accelerate the comparison process, we employ an RL sampling scheme with updating baselines.

Experimental results indicate that Rank-DSE shows superior performance in both the quality and cost of search results, with up to 7% improvement in terms of HV and 53.09% exploration cost reduction compared with the cutting-edge methods. Besides, Rank-DSE also achieves the best performance compared with the winners at the ICCAD CAD Contest 2022 and GRL-DSE. While Rank-DSE offers scalability and systematic exploration, incorporating human expertise could improve performance in the future, since automated DSE and manual design are complementary.

## References

[1] Sicheng Li, Chen Bai, Xuechao Wei, Bizhao Shi, Yen-Kuang Chen, and Yuan Xie. 2022. 2022 ICCAD CAD Contest Problem C: Microarchitecture Design Space Exploration. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. 1–7.

[2] Vinod Kathail, Shail Aditya, Robert Schreiber, B. Ramakrishna Rau, Darren C Cronquist, and Mukund Sivaraman. 2002. PICO: Automatically designing custom computers. *IEEE Transactions on Computers* 35, 9 (2002), 39–47.

[3] David Brooks, Pradip Bose, Viji Srinivasan, Michael K. Gschwind, Philip G. Emma, and Michael G. Rosenfield. 2003. New methodology for early-stage, microarchitecture-level power-performance analysis of microprocessors. *IBM Journal of Research and Development* 47, 5.6 (2003), 653–670.

[4] Christophe Dubach, Timothy Jones, and Michael O'Boyle. 2007. Microarchitectural design space exploration using an architecture-centric approach. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 262–271.

[5] Christophe Dubach, Timothy M. Jones, and Michael FP O'Boyle. 2008. Exploring and predicting the architecture/optimising compiler co-design space. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*. 31–40.

[6] Omid Azizi, Aqeel Mahesri, Benjamin C. Lee, Sanjay J. Patel, and Mark Horowitz. 2010. Energy-performance tradeoffs in processor architecture and circuit design: a marginal cost analysis. In *Proceedings of the IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 26–36.

[7] Benjamin C. Lee and David M. Brooks. 2007. Illustrative design space studies with microarchitectural regression models. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 340–351.

[8] Dandan Li, Shuzhen Yao, Yu-Hang Liu, Senzhang Wang, and Xian-He Sun. 2016. Efficient design space exploration via statistical sampling and AdaBoost learning. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[9] S. M. Shamimul Hasan, Neena Imam, Ramakrishnan Kannan, Srikanth Yoginath, and Kuldeep Kurte. 2021. Design Space Exploration of Emerging Memory Technologies for Machine Learning Applications. In *Proceedings of the International Parallel and Distributed Processing Symposium (IPDPS)*. 439–448.

[10] Chen Bai, Qi Sun, Jianwang Zhai, Yuzhe Ma, Bei Yu, and Martin DF Wong. 2021. BOOM-Explorer: RISC-V BOOM microarchitecture design space exploration framework. In *Proceedings of the 2021 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 1–9.

[11] Xiaoling Yi, Jialin Lu, Xiankui Xiong, Dong Xu, Li Shang, and Fan Yang. 2023. Graph Representation Learning for Microarchitecture Design Space Exploration. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*.

[12] Tianshi Chen, Qi Guo, Ke Tang, Olivier Temam, Zhiwei Xu, Zhi-Hua Zhou, and Yunji Chen. 2014. ArchRanker: A Ranking Approach to Design Space Exploration. In *Proceedings of the IEEE/ACM International Symposium on Computer Architecture (ISCA)*.

[13] Yixing Xu, Yunhe Wang, Kai Han, Yehui Tang, Shangling Jui, Chunjing Xu, and Chang Xu. 2021. Renas: Relativistic evaluation of neural architecture search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 4411–4420.

[14] Yaofo Chen, Yong Guo, Qi Chen, Minli Li, Wei Zeng, Yaowei Wang, and Mingkui Tan. 2021. Contrastive neural architecture search with neural architecture comparators. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 9502–9511.

[15] Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. 2007. Learning to rank: From pairwise approach to listwise approach. In *Proceedings of the International Conference on Machine Learning (ICML)*. 129–136.

[16] Marco Morik, Ashudeep Singh, Jessica Hong, and Thorsten Joachims. 2020. Controlling fairness and bias in dynamic learning-to-rank. In *Proceedings of the ACM SIGIR Conference*. 429–438.

[17] P. Li, Q. Wu, and C. Burges. 2007. Mcrank: Learning to rank using multiple classification and gradient boosting. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*.

[18] David Sculley. 2010. Combined regression and ranking. In *Proceedings of the ACM International Conference on Knowledge Discovery and Data Mining (KDD)*. 979–988.

[19] Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. 2005. Learning to rank using gradient descent. In *Proceedings of the International Conference on Machine Learning (ICML)*. 89–96.

[20] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. 2003. An efficient boosting algorithm for combining preferences. *Journal of Machine Learning Research* 4, Nov (2003), 933–969.

[21] Yunbo Cao, Jun Xu, Tie-Yan Liu, Hang Li, Yalou Huang, and Hsiao-Wuen Hon. 2006. Adapting ranking SVM to document retrieval. In *Proceedings of the ACM SIGIR Conference*. 186–193.

[22] Fen Xia, Tie-Yan Liu, Jue Wang, Wensheng Zhang, and Hang Li. 2008. Listwise approach to learning to rank: Theory and algorithm. In *Proceedings of the International Conference on Machine Learning (ICML)*. 1192–1199.

[23] Shuzi Niu, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. 2012. Top-k learning to rank: Labeling, ranking and evaluation. In *Proceedings of the ACM SIGIR Conference*. 751–760.

[24] Qingyao Ai, Keping Bi, Jiafeng Guo, and W. Bruce Croft. 2018. Learning a deep listwise context model for ranking refinement. In *Proceedings of the ACM SIGIR Conference*. 135–144.

[25] Fatih Cakir, Kun He, Xide Xia, Brian Kulis, and Stan Sclaroff. 2019. Deep metric learning to rank. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1861–1870.

[26] Tejas S. Karkhanis and James E. Smith. 2007. Automated design of application specific superscalar processors: an analytical approach. In *Proceedings of the IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 402–411.

[27] Hossein Golestani, Rathijit Sen, Vinson Young, and Gagan Gupta. 2022. Calipers: A criticality-aware framework for modeling processor performance. In *Proceedings of the IEEE/ACM International Symposium on Computer Architecture (ISCA)*. 1–14.

[28] Engin Ïpek, Sally A. McKee, Rich Caruana, Bronis R. de Supinski, and Martin Schulz. 2006. Efficiently exploring architectural design spaces via predictive modeling. *ACM International Conference on Architectural Support for Programming Languages and Operating Systems* 40, 5 (2006), 195–206.

[29] Ramapati Patra, Prasenjit Maji, Dipti Sakshi Srivastava, and Hemanta Kumar Mondal. 2024. Machine learning-driven performance assessment of network-on-chip architectures. *The Journal of Supercomputing* 80, 16 (2024), 24483–24519.

[30] Zhi-Liang Qian, Da-Cheng Juan, Paul Bogdan, Chi-Ying Tsui, Diana Marculescu, and Radu Marculescu. 2015. A support vector regression (SVR)-based latency model for network-on-chip (NoC) architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 3 (2015), 471–484.

[31] Geoffrey I. Webb, Eamonn Keogh, and Risto Miikkulainen. 2010. Naïve Bayes. *Encyclopedia of Machine Learning* 15, 1 (2010), 713–714.

[32] Yao Xiao, Shahin Nazarian, and Paul Bogdan. 2019. Self-optimizing and self-programming computing systems: A combined compiler, complex networks, and machine learning approach. *IEEE Transactions on Very Large Scale Integration Systems* 27, 6 (2019), 1416–1427.

[33] Yao Xiao, Shahin Nazarian, and Paul Bogdan. 2021. Plasticity-on-chip design: Exploiting self-similarity for data communications. *IEEE Transactions on Computers* 70, 6 (2021), 950–962.

[34] Yao Xiao, Guixiang Ma, Nesreen K. Ahmed, Mihai Capotă, Theodore L. Willke, Shahin Nazarian, and Paul Bogdan. 2023. End-to-end programmable computing systems. *Communications Engineering* 2, 1 (2023), 84.

[35] Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. 2020. Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*.

[36] Thomas N. Kipf and Max Welling. 2017. Semi-Supervised classification with graph convolutional networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*.

[37] Jianwang Zhai, Yici Cai, and Bei Yu. 2023. Microarchitecture power modeling via artificial neural network and transfer learning. In *Proceedings of the IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. 302–307.

[38] R. J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8, 3, (1992) 229–256.

[39] Jincheng Mei, Wesley Chung, Valentin Thomas, Bo Dai, Csaba Szepesvari, and Dale Schuurmans. 2022. The role of baselines in policy gradient optimization. In *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS)*.

[40] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. 2020. Monte carlo gradient estimation in machine learning. *Journal of Machine Learning Research* 21, 1 (2020), 5183–5244.

[41] Hung-Yi Liu and Luca P. Carloni. 2013. On learning-based methods for design-space exploration with high-level synthesis. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*. 1–7.

[42] Berkin Ozisikyilmaz, Gokhan Memik, and Alok Choudhary. 2008. Efficient system design space exploration using machine learning techniques. In *Proceedings of the 45th Annual Design Automation Conference*. 966–969.

[43] Robert B. Gramacy, Herbert K. H. Lee, and William G. Macready. 2004. Parameter space exploration with Gaussian process trees. In *Proceedings of the 21st International Conference on Machine Learning*. 45.

[44] Haoyi Zhang, Jiahao Song, Xiaohan Gao, Xiyuan Tang, Yibo Lin, Runsheng Wang, and Ru Huang. 2024. EasyACIM: An End-to-End automated analog CIM with synthesizable architecture and agile design space exploration. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*. 1–6.

[45] Chen Bai, Jianwang Zhai, Yuzhe Ma, Bei Yu, and Martin DF Wong. 2024. Towards automated RISC-V microarchitecture design with reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*. 12–20.

[46] Gianluca Palermo, Cristina Silvano, and Vittorio Zaccaria. 2009. ReSPIR: a response surface-based Pareto iterative refinement for application-specific design space exploration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28, 12 (2009), 1816–1829.

[47] Jerry Zhao, Ben Korpan, Abraham Gonzalez, and Krste Asanovic. 2020. Sonicboom: The 3rd generation berkeley out-of-order machine. In *Proceedings of the 4th Workshop on Computer Architecture Research with RISC-V*.

[48] Brandon Reagen, Josá Miguel Hernández-Lobato, Robert Adolf, Michael Gelbart, Paul Whatmough, Gu-Yeon Wei, and David Brooks. 2017 A case for efficient accelerator design space exploration via Bayesian optimization. In *Proceedings of the IEEE International Symposium on Low Power Electronics and Design (ISLPED)*. 1–6.

[49] Pranab Kumar Sen. 1968. Estimates of the regression coefficient based on Kendall's tau. *Journal of the American Statistical Association* 63, 324 (1968), 1379–1389.

[50] Y. Ozaki, Y. Tanigaki, S. Watanabe, M. Nomura, and M. Onishi. 2022. Multiobjective tree-structured parzen estimator. *Journal of Artificial Intelligence Research* 73 (2022), 1209–1250.

[51] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. 2012. Chisel: constructing hardware in a Scala embedded language. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*. 1216–1225.

[52] Adam Izraelevitz, Jack Koenig, Patrick Li, Richard Lin, Angie Wang, Albert Magyar, Donggyu Kim, Colin Schmidt, Chick Markley, Jim Lawson, et al. 2017. Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 209–216.

[53] Patrick S. Li, Adam M. Izraelevitz, and Jonathan Bachrach. 2016. *Specification for the FIRRTL Language*. Technical Report UCB/EECS-2016-9. EECS Department, University of California, Berkeley. Retrieved from http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-9.html

[54] UC Berkeley Architecture Research. 2018. Barstools: Useful utilities for BAR projects [Software]. Retrieved October 10, 2024, from https://github.com/ucb-bar/barstools

[55] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, and G. Yeric. 2016. ASAP7: A 7-nm finFET predictive process design kit. *Microelectronics Journal* 53 (2016), 105–115.

Received 2 December 2024; revised 28 April 2025; accepted 24 May 2025