Is Vanilla Bayesian Optimization Enough for High-Dimensional Architecture Design Optimization?

Yuanhang Gao[1,*], Donger Luo[2,*], Chen Bai[3], Bei Yu[3], Hao Geng[2], Qi Sun[1,4,#], Cheng Zhuo[1,#]
[1]Zhejiang University, [2]ShanghaiTech University, [3]Chinese University of Hong Kong
[4]State Key Laboratory of Integrated Chips and Systems

ABSTRACT

In the tide of explosive development in artificial intelligence (AI), the design of AI System-on-Chips (SoCs) is an urgently pressing issue that needs to be addressed. The application of Design Space Exploration (DSE) methods is paramount in pursuing a sound microarchitecture design and improving the quality of results. However, the high-dimensional design parameters and huge design space, which normally occur in the complicated SoCs for Large Language Model (LLM) tasks, pose a great challenge to existing techniques. In this paper, a novel and explainable Bayesian optimization-based framework MCT-Explorer is proposed. A Monte Carlo Tree Search (MCTS)-based method is utilized to analyze the importance of design parameters, guide the sampling directions, mitigate low-quality performance modeling issues, and further improve optimization efficiency. Besides, an information-guided multi-objective optimization function is adopted to balance the multiple metrics (*e.g.*, Cycle. Area, and Power) for SoC design. Our approach can provide guiding opinions and deeper insights for parameter optimization, thus transcending previous arts and achieving an explainable model. Experiment results demonstrate the extraordinary performance of our framework in various high-dimensional (up to hundreds of parameters) and complicated LLM SoC designs.

1 INTRODUCTION

The rapid evolution of artificial intelligence algorithms imposes increasingly stringent requirements on hardware catered specifically for accelerated computation. Large language models and generative AI necessitate considerable computational resources for model training and deployment [1], further reinforcing the pressing need. Designing a customized and well-optimized AI accelerator for a specific neural network takes great effort. Traditionally, achieving a high-quality design involves the expertise of professionals and a vast amount of engineering tests and revisions, usually spanning several years [2]. This is not feasible for emerging AI tasks.

Recently, some systematic, rapid, and general development processes have been proposed and adopted by academics and industry

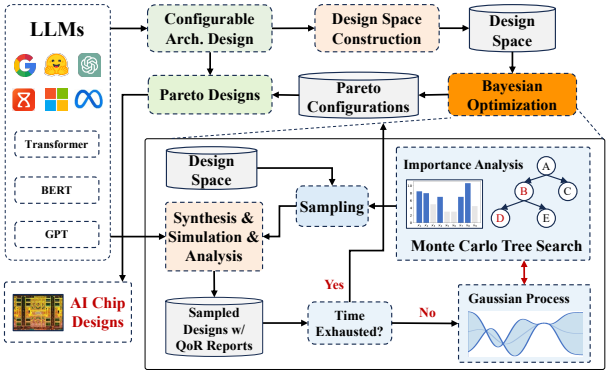

Figure 1: Overview of our MCT-Explorer.

to cut design overhead and accelerate design cycles. For example, Chipyard [3] and ESP [4] offer ease for users to complete the entire design process through configurable modules and design parameters. These design parameters create a vast design space, making the optimization process a typical design space exploration (DSE) problem. Users are empowered to explore the design space and achieve a design that caters to their specific needs by appropriately selecting and adjusting parameters.

Although many tools simplify SoC design and transform it into a process of setting parameters, identifying optimal parameter combinations from an exponentially vast space presents a significant challenge. On the one hand, the extensive design space makes it unfeasible to conduct exhaustive testing for all potential combinations. On the other hand, it is very time-intensive to synthesize a design and conduct the power, performance, and area (PPA) simulations, for up to several hours to days. Besides, the three metrics are highly correlated. Parameter adjustments may improve one metric while adversely impacting another. This dilemma in DSE presents significant obstacles to achieving an optimal design that balances each metric.

Some DSE approaches have been proposed to identify optimal parameter configurations. Ïpek *et al.* [5] use artificial neural networks to depict the connections between design parameters and estimate resulting performance. A nonlinear regression model, proposed by Lee *et al.* [6], aids in multiprocessor microarchitecture exploration. ELSE [7] introduces a fusion of various regression models to capture a reliable design space of microprocessors. Li *et al.* [8] employ the AdaBoost algorithm to explore the design space. Bai *et al.* [9] adopt the Bayesian optimization to tune the Berkeley Out-of-Order Machine (BOOM) [10]. IT-DSE [11] employs historical design data to construct a surrogate model, followed by a BO algorithm to identify an optimal solution. GRL-DSE [12] uses graph representation learning to build a compact and continuous embedding space and

* Equal contributions.
# Corresponding authors: {qisunchn, czhuo}@zju.edu.cn.

ICCAD '24, October 27–31, 2024, New York, NY, USA

ACM ISBN 979-8-4007-1077-3/24/10…$15.00
https://doi.org/10.1145/3676536.3676746

then uses an ensemble surrogate model to conduct the microarchitecture DSE. Besides, more and more approaches based on active learning are proposed [13, 14].

However, the previous arts fail to handle the high-dimensional parameters efficiently, especially for the AI SoCs for LLM-like large-scale tasks. High-dimension parameter design space, which is composed of tens to hundreds parameters and results in larger than 10^{30} parameter configurations, will not only make it harder for a surrogate model to fit the potential relationships between inputs and outputs but also give the DSE algorithm less opportunity to iterate a better solution within a few steps [15]. These approaches do not effectively identify critical parameters from among the hundreds, which are essential for providing valuable guidance in new designs.

To counteract these issues, we introduce MCT-Explorer, a Monte Carlo tree search [16] (MCTS)-based parameter analysis algorithm into the Bayesian optimization framework, as shown in Figure 1. Our framework could select critical parameters in all parameters, explore the design space in more promising directions, and provide explainable and high-quality results in fewer optimization steps. Our contributions are summarized as follows:

- A new microarchitecture design space exploration framework, MCT-Explorer, is proposed to address the high-dimensional parameter dilemma. Our framework leverages black-box optimization for high efficiency while providing explainable results.
- We utilize the Monte Carol Tree Search to select important parameters, thus mitigating the issue of inaccurate fitting in high-dimensional Bayesian Optimization.
- An information-guided multi-objective acquisition function is adopted to balance multiple design metrics of SoC design.
- We verify our framework on two open-source RISC-V SoCs and one in-house design, with 19 parameters, 65 parameters, and 270 parameters, respectively. Experimental results demonstrate the efficiency and effectiveness of MCT-Explorer compared to other state-of-the-art methods.

2 PRELIMINARIES

2.1 SoC and LLM Acceleration

System-on-chip (SoC) designs utilizing the RISC-V architecture are spearheading a revolution, credited to their remarkable efficiency and ecosystems, which facilitate the creation of versatile open-source hardware components such as Rocket [17], BOOM [18], and Gemmini [19]. The advent of SoCs specifically engineered for AI acceleration is of paramount importance. Particularly, the compute-intensive nature of large language models (LLM) is characterized by their consequential need to overcome pervasive matrix computation bottlenecks. This task hinges on the efficient designs of SoCs.

An SoC for AI accelerations integrates components such as a RISC-V CPU, specialized AI accelerators, and memory hierarchies, including Cache and DRAM. Chipyard [3], an open-source framework capable of developing Chisel-based RISC-V SoCs, represents a significant boon for developers. It integrates the aforementioned configurable processors and DNN accelerators, empowering developers to configure the various components to formulate an LLM-specific SoC.

To formulate an SoC design with high-quality power, performance, and area (PPA) tailored for LLMs, a comprehensive assessment of all configurable parameters is imperative. It determines the CPU variants, accelerator configurations, cache capacities, and *etc.* However, dynamic emerging requirements lead to complex designs with many parameters, making it difficult to achieve an optimal high-dimensional SoC design. Traditional optimization strategies suffer from the escalating number of design parameters, thus necessitating a pioneering approach in high-dimensional Design Space Exploration (DSE).

2.2 Monte Carol Tree Search

The Monte Carlo Tree Search (MCTS) algorithm [16, 20], based on random sampling, has shown great success in solving black-box optimization problems [21, 22]. This method capitalizes on the statistics from executed actions to guide the selection of subsequent steps, exhibiting superior performance in high-dimensional tasks, such as the game of Go [23]. MCTS elaborates a tree where each node denotes a specific state of the scenario at hand, each associated with an Upper Confidence Bound (UCB) [24] to assess the node's selection viability:

$$\mathrm{UCB}(X) = v_X + 2C_p\sqrt{2(\log n_p)/n_X}, \tag{1}$$

where v_X represents the expected goodness of the node, C_p is a hyper-parameter, n_p denotes the total number of times the parent node of X has been visited, and n_X is the number of times X has been visited. Considering goodness and visited frequency, UCB balances exploitation and exploration, which is crucial in black-box problem optimization.

2.3 Multi-objective Bayesian Optimization

Bayesian optimization (BO) [25] is a heuristic strategy for optimizing noisy black-box functions. It builds surrogate models to mimic the unknown black-box objective functions. Predominantly utilized as a surrogate within this domain is the Gaussian process (GP) model, which provides a probabilistic depiction of functions articulated as a posterior distribution.

When confronted with real-world optimization challenges, one often encounters scenarios with multiple, sometimes conflicting, objectives. Addressing such complexity, multi-objective Bayesian optimization (MOBO) evolves from the foundational principles of BO to deal with such problems. Let functions $\{\mathbf{f}_i(\mathbf{x})\}_{i=1}^m$, *i.e.*, $\mathbf{f}_1$ = power, $\mathbf{f}_2$ = area, $\mathbf{f}_3$ = cycles, denote the m-dimension metrics to be minimized and $\mathbb{X}$ denotes the parameter space. A parameter vector $\mathbf{x}^* \in \mathbb{X}$ is said to (Pareto) dominate $\mathbf{x}' \in \mathbb{X}$, denoted as $\mathbf{x}^* \succeq \mathbf{x}'$, if the following two conditions are met in this minimization problem:

(1) $\mathbf{f}_i(\mathbf{x}') \geq \mathbf{f}_i(\mathbf{x}^*), \forall i \in \{1, ..., m\}$,
(2) There exists at least one j such that $\mathbf{f}_j(\mathbf{x}') > \mathbf{f}_j(\mathbf{x}^*)$.

The set of parameter vectors that are not dominated by other vectors is called the Pareto-optimal set, denoted as $\mathbb{X}^*$, which is the target of MOBO:

$$\mathbb{X}^* = \{\mathbf{x}^* \in \mathbb{X} | \nexists \mathbf{x}' \in \mathbb{X}, \mathbf{x}' \succeq \mathbf{x}^*\}. \tag{2}$$

For the problem with multiple optimization objectives, Bayesian optimization builds surrogate models for these unknown black-box

objectives. Further, a multi-objective acquisition function [15, 26–28] is built based on the surrogate models and estimates the quality of a parameter configuration $\mathbf{x}$ with respect to finding the Pareto set without going through the time-consuming EDA flow.

2.4 Problem Formulation

Definition 1 (Microarchitecture Embedding)**.** Microarchitecture embedding is to define a combination of a specific SoC design's configurable parameters.

Definition 2 (Hypervolume)**.** In multi-objective problems, the solution is that of Pareto fronts. The quality of such fronts can be measured by hypervolume:

$$\mathrm{HV}\left(\mathbf{f}^{\mathrm{ref}}, \mathbf{X}\right) = \Lambda\left(\bigcup_{\mathbf{x}\in\mathbf{X}}\left[\mathbf{f}_1(\mathbf{x}), \mathbf{f}_1^{\mathrm{ref}}\right] \times \cdots \times \left[\mathbf{f}_m(\mathbf{x}), \mathbf{f}_m^{\mathrm{ref}}\right]\right), \quad (3)$$

where $\mathbf{f}^{\mathrm{ref}}$ refers to a chosen reference point which is generally set as a bad performance value point. $\mathbf{f}_m$ means the m-th metric, *e.g.*, cycles, power, area, *etc.* And Λ refers to the Lebesgue measure.

Definition 3 (ADRS)**.** Another metric to measure the quality of Pareto fronts is the average distance to reference set (ADRS):

$$\mathrm{ADRS}(\Gamma, \Omega) = \frac{1}{|\Gamma|}\sum_{\gamma\in\Gamma}\min_{\omega\in\Omega} dist(\gamma, \omega), \quad (4)$$

where $dist$ is the Euclidean distance function. Γ is the real Pareto-optimal set, *i.e.*, the reference set. Ω is the learned Pareto-optimal set. ADRS is often exploited to measure how close a learned Pareto-optimal set is to the real Pareto-optimal set of the design space.

Problem 1 (High Dimension Microarchitecture Design Space Exploration)**.** Given a search space $\mathbb{X}$, each microarchitecture design inside $\mathbb{X}$ is regarded as a feature vector $\mathbf{x}$. Metric space is $\mathbb{Y} = \{\mathbf{y} | \mathbf{y} = \mathbf{f}(\mathbf{x}), \mathbf{x} \in \mathbb{X}\}$. We define the design space exploration of SoC as finding the subset $\mathbb{X}^* \in \mathbb{X}$ with corresponding metrics $\mathbb{Y}^*$ forming the Pareto optimal set.

3 MCT-EXPLORER

3.1 Overview

Figure 1 displays the overview of our MCT-Explorer framework. Given the LLM tasks and configurable SoC designs, we determine the design space to be explored based on the architecture constraints and engineers' expertise. The exploration flow stage samples potential promising designs based on the proposed MCTS-based Bayesian optimization. We use the Gaussian process as the surrogate model and joint entropy search as the information-guided acquisition function. The EDA tool flow, including synthesis, simulation, analysis, *etc.*, is used to obtain PPA reports. The surrogate model is rebuilt with pairs of designs and their corresponding PPA values. All explored designs and their corresponding PPA values are maintained by an exploration set. The predictive Pareto optimality is acquired when a pre-determined time budget is met.

3.2 The Measure of Importance for Each Parameter

As the number of configurable parameters of SoC design increases, it is hard to construct a surrogate model to map a design

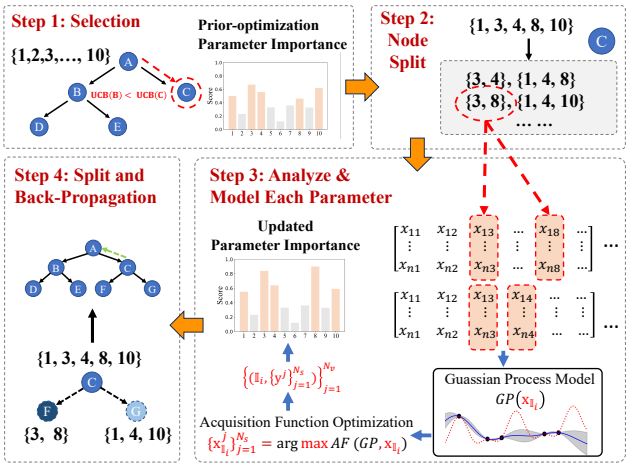

Figure 2: Overview of MCTS-based Parameter Selection.

configuration (microarchitecture embedding) to corresponding PPA values. However, we can alleviate the difficulty in the surrogate model construction, by selecting the configurable parameters according to the parameter importance. The reason behind this is based on our insight, *i.e.*, some important parameters affect the PPA values more while others contribute less to PPA values.

To quantify the importance of each parameter, we define an index set $\mathbb{D} = \{1, 2, 3, \ldots, d\}$, where d refers to the number of parameters. And we define $\mathbf{x}_{\mathbb{I}}$ as the "reduced" microarchitecture embedding, where $\mathbb{I} \subseteq \mathbb{D}$. "Reduced" means that elements of $\mathbf{x}_{\mathbb{I}}$ are selected from the original microarchitecture embedding $\mathbf{x}$ according to $\mathbb{I}$. For example, given $\mathbf{x} = (2, 4, 8, \ldots, 4)$ and $\mathbb{I} = \{2, 3\}$, $\mathbf{x}_{\mathbb{I}}$ is $(4, 8)$ as 4 and 8 are the second and the third parameters of $\mathbf{x}$ *w.r.t.* $\mathbb{I}$. We later illustrate how we obtain $\mathbb{I}$ from our algorithm in Section 3.3.

As introduced in Section 3.1, we maintain an exploration set in MCT-explorer. The exploration set stores initialized microarchitecture embeddings and corresponding PPA values evaluated from the EDA tool flow. The exploration set is gradually aggregated with newly sampled microarchitecture embeddings and PPA values in each iteration (Figure 1). To decide whether each parameter is the main contributor to the PPA values, we construct a set $\mathbb{T}$, the element of which is a pair of $(\mathbb{I}, \mathbb{M})$, where $\mathbb{M}$ is a set containing several sets of PPA values of the microarchitecture embeddings generated according to $\mathbb{I}$. A parameter score $\mathbf{s}$ is introduced to measure the importance of each parameter, as shown in Equation (5).

$$\mathbf{s} = \frac{\sum_{(\mathbb{I},\mathbb{M})\in\mathbb{T}}\sum_{\mathbf{y}\in\mathbb{M}}\mathrm{HV}(\mathbf{f}^{\mathrm{ref}}, \mathbf{y}) \cdot g(\mathbb{I})}{\sum_{(\mathbb{I},\mathbb{M})\in\mathbb{T}}|\mathbb{M}| \cdot g(\mathbb{I})} \quad (5)$$

where $\mathrm{HV}(\mathbf{f}^{\mathrm{ref}}, \mathbf{y})$ represents the hypervolume surrounded by $\mathbf{y} \in \mathbb{M}$, and a chosen reference point $\mathbf{f}^{\mathrm{ref}}$ [1]. Function g maps $\mathbb{I}$ to $\{0, 1\}^d$, an d-dimensional vector where, at the i-th position, 0 hints that the i-th parameter is not selected and 1 holds the opposite meaning. $|\mathbb{M}|$ is the number of $\mathbf{y}$ in $\mathbb{M}$, *i.e.*, how many sets of PPA values involved in $\mathbb{M}$. $\mathbf{s}$ is a vector, and the computation of Equation (5) is in an element-wise division manner.

The rationale of Equation (5) is as follows. The numerator of Equation (5) represents the sum of contributions of each parameter

[1] We reuse the symbol of HV defined in Section 2.4 with minor revision.

across all pairs $(\mathbb{I}, \mathbb{M}) \in \mathbb{T}$. If $\mathrm{HV}(\mathbf{f}^{\mathrm{ref}}, \mathbf{y})$ is larger, parameters selected according to $\mathbb{I}$ are more important since better Pareto front is achieved. The denominator of Equation (5) represents the frequency of occurrence for each parameter in $\mathbb{T}$, serving as a normalization. The i-th element of $\mathbf{s}$ is equal to the average hypervolume achieved by $\mathbf{y}$ of design configuration acquired by optimizing parameters whose index sets include i.

3.3 Monte Carol Tree Search-based Parameter Selection

Monte Carol Tree Search (MCTS) is leveraged to dynamically partition parameters into important ones and unimportant ones based on the parameter score in Section 3.2. Each node N in Monte Carol Tree (MCT) represents some parameters of microarchitecture embedding. We define the index set of parameters represented by node N as $\mathbb{A}_N \subseteq \mathbb{D}$. At each iteration of MCTS, a node X is chosen according to the Upper Confidence Bound (UCB), and parameters to be optimized are limited to the parameters involved in node X, whose indexes are contained in $\mathbb{A}_X \subseteq \mathbb{D}$. Then, the parameters in node X are re-evaluated, after which the important and unimportant ones are selected based on the updated score and assigned to the left and right child of X. Thus, parameters are further partitioned.

The overview of MCTS-based parameter selection is shown in Figure 2. The root of the tree represents all parameters. Node A is the root of the MCT in Figure 2 and $\mathbb{A}_A = \{1, 2, 3, \ldots, 10\}$ assuming the number of total parameters is 10 at this time. As described in Section 2.2, each node has a value UCB evaluated for selection according to Equation (1), in which v_X is based on $\mathbf{s}$, *i.e.*,

$$v_X = \mathbf{s} \cdot g(\mathbb{A}_X)/|\mathbb{A}_X|, \tag{6}$$

where v_X stands for the average importance of the parameters contained in node X. Given v_X, n_p, and n_X, the UCB of node X could be computed. At each iteration of MCTS, in step 1, we first select a node with a larger UCB until meeting a leaf node, denoted as node X. At the top-left of Figure 2, the MCT contains five nodes after previous iterations. At this iteration, UCB(C) is larger than UCB(B) and C is a leaf node so that node C is chosen, *i.e.*, X=C.

Next, in step 2, we aspire to divide parameters in node C into important and unimportant ones. To differentiate the importance of parameters in C more clearly, we need to make additional simulations for the parameters in C. However, it is unwise to continue optimizing all parameters in C owing to the goodness of generated design configurations contributing to all parameters in node C and not really distinguishing them. Therefore, MCT-Explorer samples N_v index subset $\mathbb{I}_i$ and corresponding complementary set $\bar{\mathbb{I}}_i = \mathbb{A}_C \setminus \mathbb{I}_i$ from $\mathbb{A}_C$ randomly for $i = 1, .., N_v$, which avoid the situation that some parameters are not lucky enough to be sampled. N_v is a pre-set hyper-parameter. For example, $\mathbb{A}_C = \{1, 3, 4, 8, 10\}$ and we randomly generate $N_v = 2$ subsets $\{3, 4\}$, $\{3, 8\}$ and their complementary set $\{1, 8, 10\}$, $\{1, 4, 10\}$ out of $\mathbb{A}_C$.

Multi-objective Bayesian Optimization (MOBO), which is later illustrated in more detail in Section 3.4, is applied in step 3 to optimize parameters whose indexes are involved in $\mathbb{I}_i$. The reduced microarchitecture embeddings (Section 3.2) of the explored design configurations, denoted as $\mathbf{X}_{\mathbb{I}_i}$ and their corresponding metric data $\mathbf{Y}$ are used to construct the surrogate model. For example, $\mathbb{I}_1 = \{3, 8\}$, and the 3-rd and 8-th parameters of all explored design configurations that are circled by light orange block is $\mathbf{X}_{\mathbb{I}_1}$. Reduced candidate embeddings $\mathbf{x}_{\mathbb{I}_i,j}$ with the j-th maximum value of the acquisition function are acquired for $j = 1, ..., N_s$, where N_s is also a pre-set hyper-parameter. Unlike optimizing all parameters, the candidate reduced embeddings gained from our BO only have parameters whose indexes are in $\mathbb{I}_i$. Since an incomplete reduced embedding cannot be input into EDA flow to get its evaluation metrics, we have to fill in the reduced embedding $\mathbf{x}_{\mathbb{I}_i,j}$ to original embedding $\mathbf{x}_j$. $\{(\mathbb{I}_i, \{\mathbf{y}_j\}_{j=1}^{N_s})\}$ is recorded into $\mathbb{T}$. After the above process in step 3 is repeated for all $\mathbb{I}_i$ and $\bar{\mathbb{I}}_i$, given the updated $\mathbb{T}$, the scores of the parameters in Node C could be updated according to Equation (5). The change in scores is evident in the comparison between prior and updated bar charts, in which the light orange bars stand for the score of parameters in node C while grey bars stand for the score of parameters outside node C.

Algorithm 1 Fill-in$(\mathbb{X}^*, \mathbb{I}, \mathbf{x}_{\mathbb{I}}, k)$

Input: $\mathbb{X}^*$ is the Pareto Set of explored design configurations, $\mathbb{I}$ is a index subset of $\mathbb{D}$, $\mathbf{x}_{\mathbb{I}}$ is a reduced microarchitecture embedding according to index set $\mathbb{I}$, and k is the number of Pareto configurations to fill in the absent part.

Output: $\mathbf{x}$: design configuration with full n-dimension parameter.

1: $\mathbb{N} = \Phi$;
2: **for** $i = 1 : k$ **do**
3: $\quad \mathbf{x}^* = \underset{\mathbf{x}' \in \mathbb{X}^* \wedge \mathbf{x}' \notin \mathbb{N}}{\arg\min} \; (dist(\mathbf{x}'_{\mathbb{I}} - \mathbf{x}_{\mathbb{I}}))$;
4: $\quad \mathbb{N} = \mathbb{N} \cup \{\mathbf{x}^*\}$;
5: **end for**
6: $\mathbf{x}_{\mathbb{D}\setminus\mathbb{I}} = \underset{\mathbf{x}' \in \mathbb{N}}{average}(\mathbf{x}'_{\mathbb{D}\setminus\mathbb{I}})$;
7: $\mathbf{x} = combine(\mathbf{x}_{\mathbb{I}}, \mathbf{x}_{\mathbb{D}\setminus\mathbb{I}})$;
8: **return** $\mathbf{x}$;

The proposed fill-in approach is depicted in Algorithm 1. We seek in the current Pareto set for the nearest k Pareto configurations based on the parameter index set $\mathbb{I}$ and average the absent part , *i.e.*, the parameters indexed by $\mathbb{D} \setminus \mathbb{I}$, of these k Pareto configurations as the absent part of our candidate reduced embedding. The complete design configuration $\mathbf{x}$ is combined by the reduced embedding $\mathbf{x}_{\mathbb{I}}$ and the absent part $\mathbf{x}_{\mathbb{D}\setminus\mathbb{I}}$. The filled-in $\mathbf{x}$ may not fall within the allowed design space. In such a situation, we can search for the closest possible alternative within all the available design options.

Finally, in step 4, parameters in the current node are divided into two child nodes based on the updated scores. The more important ones are split into the left child while the others are split into the right child. In the updated importance score bar chart at the middle of Figure 2, the 3-rd and 8-th parameters score higher. Thus the 3-rd, 8-th parameters are divided into node F, the left child of C, while the 1-st, 4-th and 10-th are divided into node G . Besides, the new scores will change the UCB of ancestor nodes of the current node C, which also includes the current node's parameters. Therefore, as indicated by the green arrow in Figure 2, we back-propagate to update the UCB of ancestor nodes.

The detailed algorithm of Node-Analysis is presented as Algorithm 2. lines 3-15 correspond to the steps 2-4 of Figure 2.

Algorithm 2 Node-Analysis$(X, \mathbb{T}, \mathbb{X}^*, N_v, N_s, k)$

Input: X is the selected node, $\mathbb{T}$ is information set, $\mathbb{X}^*$ is current pareto set, N_v is the batch size of parameter index subsets, N_s is the sample batch size, k is the number of Pareto configurations to fill in the absent part.

Output: T_X: The simulation time cost by EDA_flow, $\mathbb{T}$: the updated information set.

1: $\mathbb{A}_X$ = indexes of the subset of parameters represented by X;
2: $T_X = 0$;
3: **for** $i = 1 : N_v$ **do**
4: $\mathbb{I}_i$= sampled index subset from $\mathbb{A}_X$;
5: $\mathbf{X}_{\mathbb{I}_i}$ is reduced microarchitecture embeddings of explored design configurations and $\mathbf{Y}$ is the corresponding metric data;
6: Fit a group of GP models using $\mathbf{X}_{\mathbb{I}_i}$ and $\mathbf{Y}$;
7: Sample reduced embedding $\mathbf{x}_{\mathbb{I}_i,j}$ with the j-th maximum value of the acquisition function for $j = 1, ..., N_s$;
8: $\mathbf{x}_j$ =Full-in($\mathbb{X}^*, \mathbb{I}_i, \mathbf{x}_{\mathbb{I}_i,j}, k$) for $j = 1, 2, \cdots, N_s$; ▷ algorithm 1
9: $\mathbf{y}_j$ = EDA_flow($\mathbf{x}_j$) for $j = 1, 2, \cdots, N_s$;
10: $T_X = T_X$ + runtime_overhead of EDA_flow;
11: $\mathbb{T} = \mathbb{T} \cup \{(\mathbb{I}_i, \{\mathbf{y}_j\}_{j=1}^{N_s})\}$;
12: Repeat lines 5-11 for $\bar{\mathbb{I}}_i = \mathbb{A}_X \setminus \mathbb{I}_i$;
13: **end for**
14: Calculate the parameter score s using $\mathbb{T}$; ▷ Equation (5)
15: Bifurcate the leaf node X into two child nodes;
16: **return** T_X, $\mathbb{T}$;

Algorithm 3 MCT-Explorer$(N_v, N_s, k, T, \mathbb{X}, \mathbb{D})$

Input: N_v is the batch size of parameter index subsets, N_s is the sample batch size, k is the number of Pareto configurations to fill in the absent part, T is the total time budget for EDA_flow, $\mathbb{X}$ is the design space, $\mathbb{D}$ is the index set of SoC parameters.

Output: $\mathbb{X}^*$: Pareto-optimal designs.

1: **for** $i = 1 : N_v$ **do**
2: $\mathbb{I}_i$ = sampled index subset from $\mathbb{D}$;
3: $\bar{\mathbb{I}}_i = \mathbb{D} \setminus \mathbb{I}_i$;
4: $\mathbb{X}_i$= $\{\mathbf{x}_j\}_{j=1}^{N_s}$ sampled from $\mathbb{X}$; $\bar{\mathbb{X}}_i$= $\{\mathbf{x}_j\}_{j=1}^{N_s}$ sampled from $\mathbb{X}$;
5: $\mathbb{M}_i$= EDA_flow($\mathbb{X}_i$); $\bar{\mathbb{M}}_i$= EDA_flow($\bar{\mathbb{X}}_i$);
6: $T = T-$ runtime_overhead of EDA_flow ;
7: **end for**
8: $\mathbb{T} = \{(\mathbb{I}_i, \mathbb{M}_i), (\bar{\mathbb{I}}_i, \bar{\mathbb{M}}_i)\}_{i=1}^{N_v}$;
9: $\mathbb{X}^*$= Current Pareto Set;
10: Calculate the parameter score $\mathbf{s}$ using $\mathbb{T}$ by Equation (5);
11: Initialize the Monte Carlo Tree;
12: **while** $T > 0$ **do**
13: X = the leaf node selected by UCB;
14: $T_X, \mathbb{T}$ = Node-Analysis($X, \mathbb{T}, \mathbb{X}^*, N_v, N_s, k$); ▷ Algorithm 2
15: $T = T - T_X$;
16: $\mathbb{X}^*$ = Updated Pareto Set;
17: Back-propagate to update the UCB value of ancestor node;
18: **end while**
19: **return** Pareto-optimal designs $\mathbb{X}^*$;

The complete MCT-Explorer is shown as Algorithm 3. In lines 1-11, it initializes the Monte Carol Tree. Firstly, it samples N_v parameter index set $\mathbb{I}_i$ from $\mathbb{D}$ for $i = 1, .., N_v$. For each $\mathbb{I}_i$ and its complementary set $\bar{\mathbb{I}}_i = \mathbb{D} \setminus \mathbb{I}_i$, we sample N_s design configurations. The initialized $\mathbb{T} = \{(\mathbb{I}_i, \mathbb{M}_i), (\bar{\mathbb{I}}_i, \bar{\mathbb{M}}_i)\}_{i=1}^{N_v}$, where $\mathbb{M}_i$ or $\bar{\mathbb{M}}_i$ is a set including several sets of PPA values of N_s sampled design configurations. The MCT is initialized with a root node, and the initial value $\mathbf{s}$ is calculated by Equation (5). In each iteration (lines 13-17), a leaf node is selected according to UCB. The parameters in the chosen leaf node X are analyzed more according to Algorithm 2. Finally, the algorithm returns the learned Pareto optimal set.

By utilizing MCT, the important parameters are partitioned into some nodes and play a crucial role in finding the next candidate design configuration, thus making the process of DSE more efficient.

3.4 Multi-Objective Exploration w. Bayesian Optimization

We build a surrogate model based on sampled designs and their corresponding PPA values evaluated from the EDA flow. The surrogate model characterizes the relation between the reduced microarchitecture embedding and PPA values. In our framework, we design the surrogate model based on the Gaussian process(GP) due to the promising results for general problems [29, 30].

Assuming that we acquire the reduced microarchitecture embeddings according to $\mathbb{I}$ of explored design configurations, *i.e.*, $\mathbf{X}_{\mathbb{I}} = [\mathbf{x}_{\mathbb{I},1}, \mathbf{x}_{\mathbb{I},2}, \ldots, \mathbf{x}_{\mathbb{I},n}]$ and its PPA values $\mathbf{Y}$, we build a group of GP models to build the latent relationship between $\mathbf{X}_{\mathbb{I}}$ and $\mathbf{Y}$. In the Gaussian Process, the value function of single metric $f(\mathbf{x}'_{\mathbb{I}})$ is given a prior with $f(\mathbf{x}'_{\mathbb{I}}) \sim \mathcal{GP}(f, \mu, k_\theta)$, where μ is the mean value and the kernel function k is parameterized by θ. The Gaussian Process could be constructed:

$$[f(\mathbf{x}_{\mathbb{I},1}), f(\mathbf{x}_{\mathbb{I},2}), \cdots, f(\mathbf{x}_{\mathbb{I},n})]^T \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{K}_{\mathbf{X}_{\mathbb{I}}\mathbf{X}_{\mathbb{I}}|\theta}), \tag{7}$$

where $\mathbf{K}_{\mathbf{X}_{\mathbb{I}}\mathbf{X}_{\mathbb{I}}|\theta}$ is the intra-covariance matrix among all vectors and can be computed via $[\mathbf{K}_{\mathbf{X}_{\mathbb{I}}\mathbf{X}_{\mathbb{I}}|\theta}]_{ij} = k_\theta(\mathbf{x}_{\mathbb{I},i}, \mathbf{x}_{\mathbb{I},j})$. To model the uncertainties of GP models, a Gaussian noise $\mathcal{N}(f(\mathbf{x}_{\mathbb{I}}), \sigma_e^2)$ is added, where σ_e^2 is the variance of noise. Given a new arbitrary reduced embedding $\mathbf{x}'_{\mathbb{I}}$, the predictive joint probability of f' based on $\mathbf{y}$, could be calculated by Equation (8). To fit the latent relationship well, the parameter θ is optimized to maximize the likelihood of the event that $\mathbf{X}_{\mathbb{I}}$ and $\mathbf{Y}$ happened together.

$$\begin{bmatrix} \mathbf{y} \\ f' \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu} \\ \mu_* \end{bmatrix}, \begin{bmatrix} \mathbf{K}_{\mathbf{X}_{\mathbb{I}}\mathbf{X}_{\mathbb{I}}|\theta} + \sigma_e^2\mathbf{I} & \mathbf{K}_{\mathbf{X}_{\mathbb{I}}\mathbf{x}'_{\mathbb{I}}|\theta} \\ \mathbf{K}_{\mathbf{x}'_{\mathbb{I}}\mathbf{X}_{\mathbb{I}}|\theta} & k_{\mathbf{x}'_{\mathbb{I}}\mathbf{x}'_{\mathbb{I}}|\theta} \end{bmatrix} \right). \tag{8}$$

The selection of the next candidate is significant. Which reduced embedding to be selected at the current iteration depends on the surrogate model and acquisition function. Due to the $\mathbf{X}_{\mathbb{I}}$ to fit GP models is not the total parameter of our SoC, each time the GP we built is not the real relationship between the full design configuration and its PPA values. We do not expect to achieve the maximum expected hypervolume increase but rather to acquire more information about the unknown search region. We adopt the information-based acquisition function $I(\mathbf{x}')$ expressed with

mutual information $MI(\cdot)$ and entropy $H(\cdot)$ as follows:

$$\begin{aligned} I(\mathbf{x}'_{\mathbb{I}}|\mathbf{X}_{\mathbb{I}},\mathbf{Y}) =& MI(\mathbf{y};(\mathbb{X}^*_{\mathbb{I}},\mathbb{Y}^*)|\mathbf{x}'_{\mathbb{I}},\mathbf{X}_{\mathbb{I}},\mathbf{Y}) \\ =& H[p(\mathbf{y}|\mathbf{x}'_{\mathbb{I}},\mathbf{X}_{\mathbb{I}},\mathbf{Y})] \\ & - \mathbb{E}_{p((\mathbb{X}^*_{\mathbb{I}},\mathbb{Y}^*)|\mathbf{X}_{\mathbb{I}},\mathbf{Y})}[H[p(\mathbf{y}|\mathbf{x}'_{\mathbb{I}},\mathbf{X}_{\mathbb{I}},\mathbf{Y},\mathbb{X}^*_{\mathbb{I}},\mathbb{Y}^*)]] \\ \approx& H[p(\mathbf{y}|\mathbf{x}'_{\mathbb{I}},\mathbf{X}_{\mathbb{I}},\mathbf{Y})] \\ & - \frac{1}{S}\sum_{s=1}^{S} H[p(\mathbf{y}|\mathbf{x}'_{\mathbb{I}},\mathbf{X}_{\mathbb{I}},\mathbf{Y},\mathbb{X}^*_{\mathbb{I},s},\mathbb{Y}^*_s)], \end{aligned} \tag{9}$$

where $\mathbf{y}$ is the PPA values of the embedding , $\mathbb{X}^*_{\mathbb{I}}$ is the reduced microarchitecture embeddings of the ground-truth Pareto optimal set of design configurations, $\mathbb{Y}^*$ is the ground-truth Pareto optimal set of metrics. Since the ground-truth Pareto optimal is unknown, to compute Equation (9), we leverage Monte-Carlo sampling to get the approximate Pareto set $\mathbb{X}^*_{\mathbb{I},s}$ and $\mathbb{Y}^*_s$. S is the number of samples. Thus, we could determine the next reduced microarchitecture embedding $\mathbf{x}^*_{\mathbb{I}}$ that maximizes Equation (9) as follows:

$$\mathbf{x}^*_{\mathbb{I}} = \arg\max_{\mathbf{x}_{\mathbb{I}}} I(\mathbf{x}_{\mathbb{I}}), \tag{10}$$

where $\mathbf{x}^*_{\mathbb{I}}$ brings the most information gain of the unexplored design space.

4 EXPERIMENTS

4.1 Experiments Settings

Comprehensive experiments have been conducted to validate the effectiveness of our proposed MCT-Explorer. Three designs are evaluated: two open-source RISC-V-based LLM SoCs and an in-house design featuring 270 parameters. The RISC-V SoCs are implemented based on Chipyard, one with 19 parameters containing a Gemmini accelerator (denoted as "Single-Gemmini") and one with 65 parameters containing two Gemmini accelerators (denoted as "Dual-Gemmini"). Chipyard [3] framework is leveraged to compile RTL designs and run simulations. We use 7-nm ASAP7 PDK [31] for the VLSI flow. RTL code of SoC is generated from parameterized Chisel files and then put into Cadence Genus 201 for synthesis to get the area overhead. Power consumption is analyzed by Cadence Joules 201 and PrimeTime 2018.06-SP1. A bespoke simulator built on Spike is utilized to get the accurate cycle report for LLM inference tasks. The LLM inference tasks include Transformer-small [32], Transformer-large, Bert-base [33], Bert-large, and GPT1-small [34]. The optimization objectives include power, cycle, and area. We compare our methods with some representative baselines:

- GLSVLSI'07 [35]: Support vector machine-based method.
- HPCA'07 [6]: Regression-based model with non-linear transformation.
- DAC'16 [8]: Method based on AdaBoost learning.
- ASPDAC'20 [36]: XGBoost-based method.
- ICCAD'21 [9]: Method with active learning-based initial dataset sampling and Gaussian process with deep kernel learning.
- ISCA'23 [37]: Method based on randomized decision forests.
- AAAI'24 [38]: Method based on reinforcement learning.

Figure 3 shows an example of the Dual-Gemmini RISC-V SoC design architecture and some parameters are provided in Table 1. *cpu_type* specifies the type of CPU cores selected from different BOOM and Rocket cores. *L2TLBs* assigns the number of entries of

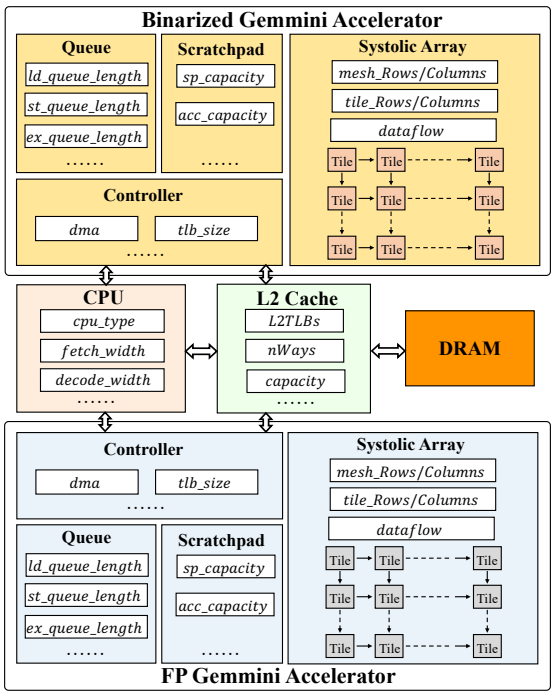

Figure 3: The architecture of the Dual-Gemmini SoC. These two accelerators share a similar architecture with different implementation details. The binarized one is customized for binarized operations with data packed as Int type.

translation look-aside buffer in L2 Cache. *nWays* assigns the number of associate sets of L2 Cache. *capacityKB* specifies the capacity of L2 Cache. *tile_Rows/Columns* and *mesh_Rows/Columns* specify the dimensions of systolic arrays. Each tile is an array of processing engines, and each mesh is an array of tiles with connecting registers. *sp_capacity* determines the capacity of scratchpad memory. The high-level structure of binarized Gemmini is similar to the general Floating-point Gemmini while the internal computation modules are customized for binarized operations, such as quantization, packing data into integers, bitwise operations, pop-count for multiplications, and de-quantization, *etc.*

For the convenience of experimental comparisons, we construct three offline datasets. The Single-Gemmini dataset contains 1124 parameter configurations. The Dual-Gemmini dataset contains 1035 configurations, and our in-house dataset contains 1300 configurations. Each dataset consumes 3000 to 5000 CPU hours to run the syntheses and simulations to get the PPA values. Note that in the experimental results, the "Runtime" includes the runtimes of algorithms and EDA flow. All experiments are conducted on a Linux server with Intel Xeon CPU (Platinum 8475B) and 512 GB RAM.

4.2 Single-Gemmini SoC Optimization

The comparison results are measured by the following metrics: hypervolume (HV), cycles-power hypervolume ($HV_{0,1}$), cycles-area hypervolume ($HV_{0,2}$), and average distance to reference Pareto-Optimal set (ADRS). The hypervolume is the higher the better while the ADRS is the vice versa. Figure 4 show the increase of hypervolume and the decrease of ADRS of each method. Table 2

Table 1: Examples of Parameters of Dual-Gemmini SoC.

Parameters	Module	Candidates	Importance [+]
cpu_type	CPU core	BOOM/Rocket	★
L2TLBs	L2 Cache	512,1024	△
nWays		4,8	★
capacityKB		512,1024	
int_tile_Rows/Columns	Binarized Acc.	1, 2	
int_mesh_Rows/Columns		8, 16	
int_tlbsize		4, 16	
int_dataflow		BOTH, WS, OS	★
int_res_ld/st/ex		4, 8, 16	
int_ld/st/ex_queue_length		4, 8, 16	★
sp_capacity		128, 256	
acc_capacity		64, 128	★
fp_tile_Rows/Columns	FP Acc.	1, 2	
fp_mesh_Rows/Columns		8, 16	
fp_tlbsize		4, 16	
fp_dataflow		BOTH, WS, OS	
fp_res_ld/st/ex		4, 8, 16	
fp_ld/st/ex_queue_length		4, 8, 16	△

[+] Parameter importance learned by our method: "★" denotes the two most important parameters and "△" denotes the least important.

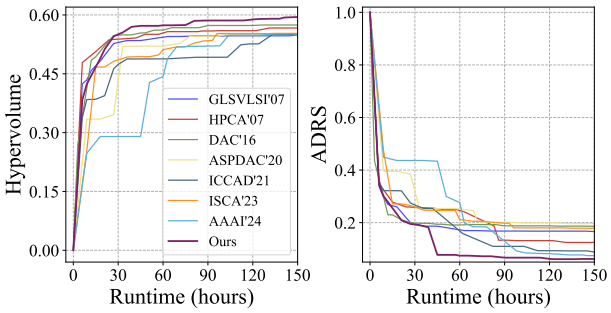

Figure 4: Comparisons on the Single-Gemmini RISC-V SoC.

shows the detail result values. On average of these three metrics, our method is 6.15% higher than GLSVLSI'07 [35], 5.15% higher than HPCA'07 [6], 5.15% higher than DAC'16 [8], 7.62% higher than ASPDAC'20 [36], 6.00% higher than ICCAD'21 [9], 6.47% higher than ISCA'23 [37], 4.82% higher than AAAI'24 [38]. Besides, our method converges much faster than the baselines. In ADRS, our method converges to 0.077 after 45 hours of runtime, while AAAI'24 [38] converges to 0.092 after 98 hours of runtime, which indicates our method's efficiency in design space exploration. The results of language model tests on Pareto designs given by each method are shown in Figure 8, in which the orange bars stand for cycles of Pareto designs output by other methods relative to our method on Single-Gemmini. The SoC design output by our method outperforms others. Concretely, the cycles to finish LLM tasks on Pareto designs output by MCT-Explorer is 25.97% lower than GLSVLSI'07 [35], 5.02% lower than HPCA'07 [6], 18.20% lower than DAC'16 [8], 13.10% lower than ASPDAC'20 [36], 9.52% lower than ICCAD'21 [9], 5.02% lower than ISCA'23 [37], 5.02% lower than AAAI'24 [38]. The results have proven the validity of our method in getting a relatively optimal design for AI acceleration.

4.3 Dual-Gemmini SoC Optimization

Table 3 shows detail results. On average of these three metrics, our method is 11.61% higher than GLSVLSI'07 [35], 6.32% higher

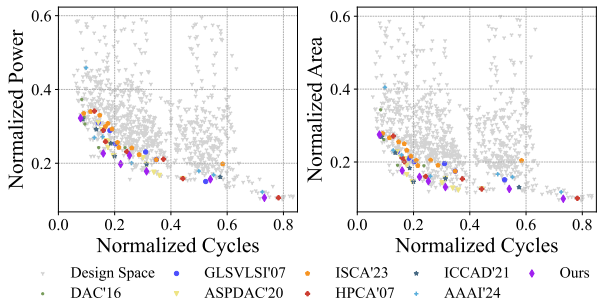

Figure 6: Learned Pareto-optimal set of Dual-Gemmini in cycles-power & cycles-area metric space.

than HPCA'07 [6], 11.81% higher than DAC'16 [8], 7.39% higher than ASPDAC'20 [36], 7.27% higher than ICCAD'21 [9], 9.75% higher than ISCA'23 [37], 5.60% higher than AAAI'24 [38]. Figure 5 shows the increase of hypervolume and the decrease of ADRS of each method. The purple curve rises faster than others in hypervolume and falls faster than others in ADRS. In ADRS, our method converges to 0.056 after 36 hours of runtime, while AAAI'24 [38] converges to 0.081 after 108 hours of runtime, which indicates our method's efficiency in design space exploration. The Pareto frontier given by each method is visualized in two QoR metrics as shown in Figure 6.

The importance scores of some representative parameters are in Figure 7. Our method prioritizes *int_dataflow*, *int_queue_ld*, *cpu_type*, *acc_capacity*, and *nWays* suggesting that optimizing these could enhance the Pareto design configuration. Conversely, *L2TLBs* and *fp_queue_ld*, as listed in Table 1, are deemed less critical by our method due to their minimal impact on the selection of the next candidate.

The blue bars in Figure 8 show the cycles of Pareto designs output by other methods relative to our method on Dual-Gemmini. The cycles to finish LLM tasks on Pareto designs output by MCT-Explorer is 19.59% lower than GLSVLSI'07 [35], 16.00% lower than HPCA'07 [6], 7.86% lower than DAC'16 [8], 19.59% lower than ASPDAC'20 [36], 16.00% lower than ICCAD'21 [9], 12.12% lower than ISCA'23 [37], 7.08% lower than AAAI'24 [38], which presents the outstanding effectiveness of our method in getting a relatively optimal design for AI acceleration.

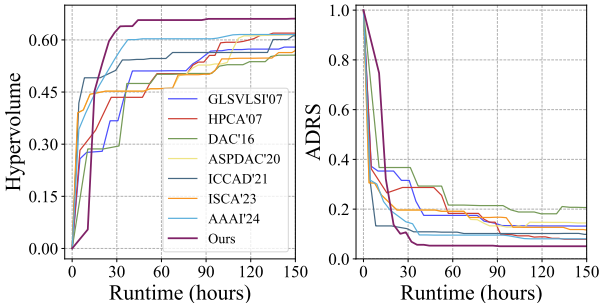

Figure 5: Comparisons on the Dual-Gemmini RISC-V SoC.

Table 2: Comparisons on the Single-Gemmini RISC-V SoC

Metric \ Method	GLSVLSI'07 [35]	HPCA'07 [6]	DAC'16 [8]	ASPDAC'20 [36]	ICCAD'21 [9]	ISCA'23 [37]	AAAI'24 [38]	**Ours**
HV_0, 1	0.7076	0.6978	0.6964	0.6882	0.7029	0.7050	0.7093	**0.7376**
HV_0, 2	0.6395	0.6500	0.6433	0.6328	0.6426	0.6330	0.6592	**0.6807**
HV	0.5496	0.5667	0.5744	0.5500	0.5538	0.5530	0.5521	**0.5950**
Average	0.6322	0.6382	0.6380	0.6236	0.6331	0.6303	0.6402	**0.6711**
Ratio(%)	94.20	95.10	95.10	92.92	94.34	93.92	95.40	**100**

Table 3: Comparisons on the Dual-Gemmini RISC-V SoC

Metric \ Method	GLSVLSI'07 [35]	HPCA'07 [6]	DAC'16 [8]	ASPDAC'20 [36]	ICCAD'21 [9]	ISCA'23 [37]	AAAI'24 [38]	**Ours**
HV_0, 1	0.7134	0.7504	0.7381	0.7453	0.7468	0.7392	0.7536	**0.7938**
HV_0, 2	0.6958	0.7176	0.6904	0.7067	0.7104	0.7072	0.7324	**0.7642**
HV	0.5793	0.6195	0.5562	0.6143	0.6118	0.5754	0.6155	**0.6611**
Average	0.6628	0.6958	0.6616	0.6888	0.6895	0.6740	0.7005	**0.7397**
Ratio(%)	89.60	94.06	89.44	93.12	93.22	91.12	94.70	**100**

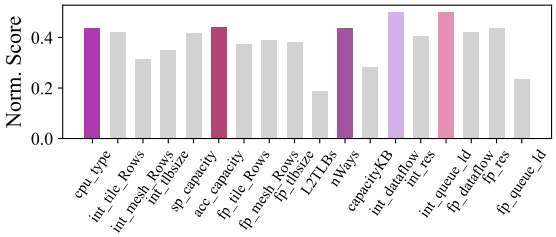

Figure 7: Importance scores of parameters in the Dual-Gemmini SoC.

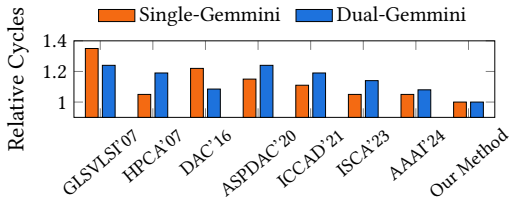

Figure 8: Comparisons on the normalized cycles of LLM tasks on the found Pareto-optimal parameter configurations.

4.4 In-house Design Optimization

As shown in Figure 9, our proposed method reaches the best objective with only 13.89% runtime overhead versus other state-of-the-art approaches. The purple line in Figure 9 rises the fastest and exceeds others early. BOOM-Explorer [9] gets a relatively high value due to its special initialization algorithm using TED [39] firstly but is soon caught up by our method. ICCAD'21 [9], GLSVLSI'07 [35], and DAC'16 [8] reach the best objective costing 7.2×, 8.2×, 10.1× runtime of our method respectively, other approaches have not reached the best objective in the limited time. After the time is exhausted, the objective our method reached is 47.84% higher than AAAI'24 [38], 53.94% higher than HPCA'07 [6], 71.84% higher than ISCA'23 [37]. Figure 10 illustrates the importance scores that help distinguish these parameters explicitly. Most parameters have a limited impact on performance, with only 26 out of 270 scoring above 0.6. These scores highlight critical directions for our optimization and offer precious analytical guidance to architects. In very high dimensions, our method outperforms all other methods significantly.

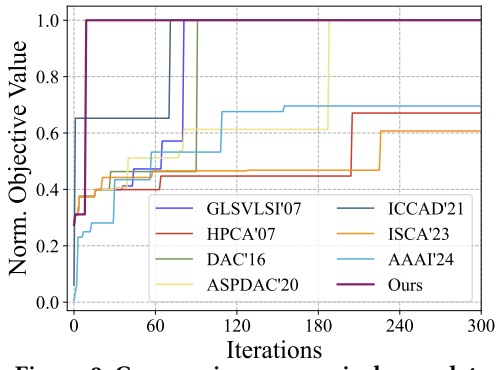

Figure 9: Comparsions on our in-house data.

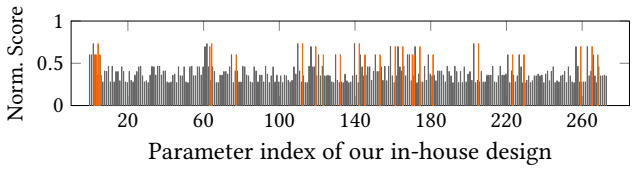

Figure 10: The importance scores of in-house design analyzed by our method. Orange represents scores above 0.6.

5 CONCLUSION

In this paper, we propose a novel Bayesian optimization framework based on the Monte Carlo Tree Search (MCTS). The MCTS-based method provides importance analyses for high-dimensional parameters while an information-guided multi-objective optimization function is adopted to balance the multiple optimization objectives. Guiding opinions and deeper insights are provided by our approach for parameter optimization, which is explainable and efficient for the DSE problem. Experiments on large language models in various SoC designs have demonstrated the extraordinary performance of the proposed framework.

ACKNOWLEDGMENTS

This work is supported by the State Key Laboratory of Integrated Chips and Systems (No. SKLICS-K202313).

REFERENCES

[1] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong *et al.*, "A survey of large language models," *arXiv preprint arXiv:2303.18223*, 2023.

[2] L.-T. Wang, Y.-W. Chang, and K.-T. T. Cheng, *Electronic design automation: synthesis, verification, and test.* Morgan Kaufmann, 2009.

[3] A. Amid *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom socs," *IEEE Micro*, 2020.

[4] P. Mantovani, D. Giri, G. Di Guglielmo, L. Piccolboni, J. Zuckerman, E. G. Cota, M. Petracca, C. Pilato, and L. P. Carloni, "Agile soc development with open esp," in *Proceedings of the 39th International Conference on Computer-Aided Design*, 2020, pp. 1–9.

[5] E. Ïpek *et al.*, "Efficiently exploring architectural design spaces via predictive modeling," in *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2006.

[6] B. C. Lee and D. M. Brooks, "Illustrative design space studies with microarchitectural regression models," in *IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2007, pp. 340–351.

[7] Q. Guo *et al.*, "Robust design space modeling," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2015.

[8] D. Li *et al.*, "Efficient design space exploration via statistical sampling and adaboost learning," in *ACM/IEEE Design Automation Conference (DAC)*, 2016.

[9] C. Bai *et al.*, "Boom-explorer: Risc-v boom microarchitecture design space exploration framework," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2021.

[10] K. Asanovic, D. A. Patterson, and C. Celio, "The berkeley out-of-order machine (BOOM): An industry-competitive, synthesizable, parameterized RISC-V processor," University of California at Berkeley, Tech. Rep., 2015.

[11] Z. Yu, C. Bail, S. Hu, R. Chen, T. He, M. Yuan, B. Yu, and M. Wong, "It-dse: Invariance risk minimized transfer microarchitecture design space exploration," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.

[12] X. Yi, J. Lu, X. Xiong, D. Xu, L. Shang, and F. Yang, "Graph representation learning for microarchitecture design space exploration," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[13] S. Chen, S. Zheng, C. Bai, W. Zhao, S. Yin, Y. Bai, and B. Yu, "Soc-tuner: An importance-guided exploration framework for dnn-targeting soc design," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 207–212.

[14] S. Huang, Y. Ye, H. Yan, and L. Shi, "Ars-flow: A design space exploration flow for accelerator-rich system based on active learning," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 213–218.

[15] F. Karl, T. Pielok, J. Moosbauer, F. Pfisterer, S. Coors, M. Binder, L. Schneider, J. Thomas, J. Richter, M. Lang *et al.*, "Multi-objective hyperparameter optimization–an overview," *arXiv preprint arXiv:2206.07438*, 2022.

[16] M. Świechowski, K. Godlewski, B. Sawicki, and J. Mańdziuk, "Monte carlo tree search: A review of recent modifications and applications," *Artificial Intelligence Review*, vol. 56, no. 3, pp. 2497–2562, 2023.

[17] K. Asanovic *et al.*, "The rocket chip generator," *EECS Department, University of California, Berkeley, Tech. Rep.*, 2016.

[18] ——, "The berkeley out-of-order machine (boom): An industry-competitive, synthesizable, parameterized risc-v processor," *University of California at Berkeley Berkeley United States, Tech. Rep*, 2015.

[19] H. Genc *et al.*, "Gemmini: Enabling systematic deep-learning architecture evaluation via full-stack integration," in *ACM/IEEE Design Automation Conference (DAC)*, 2021.

[20] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence & Ai in Games*, vol. 4, no. 1, pp. 1–43, 2012.

[21] L. Wang, R. Fonseca, and Y. Tian, "Learning search space partition for black-box optimization using monte carlo tree search," *Advances in Neural Information Processing Systems*, vol. 33, pp. 19 511–19 522, 2020.

[22] L. Song, K. Xue, X. Huang, and C. Qian, "Monte carlo tree search based variable selection for high dimensional bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 35, pp. 28 488–28 501, 2022.

[23] S. Gelly and D. Silver, "Monte-carlo tree search and rapid action value estimation in computer go," *Artificial Intelligence*, vol. 175, no. 11, pp. 1856–1875, 2011.

[24] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Machine learning*, vol. 47, pp. 235–256, 2002.

[25] J. Mockus, "The application of bayesian methods for seeking the extremum," *Towards global optimization*, 1998.

[26] D. Hernández-Lobato, J. Hernandez-Lobato, A. Shah, and R. Adams, "Predictive entropy search for multi-objective bayesian optimization," in *International conference on machine learning*. PMLR, 2016, pp. 1492–1501.

[27] S. Belakaria, A. Deshwal, and J. R. Doppa, "Max-value entropy search for multi-objective bayesian optimization," *Advances in neural information processing systems*, vol. 32, 2019.

[28] B. Tu, A. Gandy, N. Kantas, and B. Shafei, "Joint entropy search for multi-objective bayesian optimization," *Advances in Neural Information Processing Systems*, vol. 35, pp. 9922–9938, 2022.

[29] A. J. Smola and B. Schölkopf, *Learning with kernels*. Citeseer, 1998, vol. 4.

[30] C. K. Williams and C. E. Rasmussen, *Gaussian Processes for Machine Learning*. MIT press Cambridge, MA, 2006, vol. 2, no. 3.

[31] V. Vashishtha, M. Vangala, and L. T. Clark, "Asap7 predictive design kit development and cell design technology co-optimization," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 992–998.

[32] A. Vaswani *et al.*, "Attention is all you need," *Annual Conference on Neural Information Processing Systems (NeurIPS)*, 2017.

[33] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[34] A. Radford *et al.*, "Improving language understanding by generative pre-training," https://openai.com/research/language-unsupervised/, 2018.

[35] M. Barros *et al.*, "Ga-svm feasibility model and optimization kernel applied to analog ic design automation," in *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2007.

[36] Z. Xie *et al.*, "Fist: A feature-importance sampling and tree-based method for automatic design flow parameter tuning," in *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*. IEEE, 2020, pp. 19–25.

[37] S. Krishnan *et al.*, "Archgym: An open-source gymnasium for machine learning assisted architecture design," in *IEEE/ACM International Symposium on Computer Architecture (ISCA)*, 2023.

[38] C. Bai, J. Zhai, Y. Ma, B. Yu, and M. D. F. Wong, "Towards automated risc-v microarchitecture design with reinforcement learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024.

[39] K. Yu, J. Bi, and V. Tresp, "Active Learning via Transductive Experimental Design," in *International Conference on Machine Learning (ICML)*, 2006, pp. 1081–1088.