MicroEvo: Knowledge-Guided LLM Sampling for Efficient Microarchitecture Design Space Exploration

Jia Xiong[†1,2], Runkai Li[†1,2], Chenxu Niu[3], Guangyuan Gao[1,2], Changwen Xing[1,2], Yifan Zhang[1,2], Xinlai Wan[1,2], Jieran Cui[1,2], Chen Bai[4], Yusheng Hua[5], Ying Wang[6], Ming Ling[1], Xi Wang[*1,2], Tao Xie[1,7,4]
[1] Southeast University [2] National Center of Technology Innovation for EDA
[3] NVIDIA Corporation [4] Fudan University [5] Nanjing University of Posts and Telecommunications
[6] Institute of Computing Technology, Chinese Academy of Sciences [7] Peking University & Beijing Tongming Lake Center

Abstract

Microarchitecture design space exploration suffers from expansive search spaces and expensive PPA evaluation, leaving only a small simulation budget for design decision-making. Existing methods perform blind search without considering microarchitectural dependencies and fail to learn from the iterative search effectively, leading to wasted evaluations and weak Pareto convergence. In this paper, we propose **MicroEvo**, a knowledge-guided framework that couples off-the-shelf LLMs with Monte Carlo Tree Search (MCTS) for multi-objective microarchitecture optimization. MicroEvo combines LLM-driven evolutionary operators, a Pareto-aware tree policy that balances Pareto contribution and diversity, an active knowledge accumulation mechanism that extracts and reuses optimization insights, and state-aware directives that adapt the search behavior online. Experiments show that MicroEvo improves Pareto-front quality by up to **36.2%** over NSGA-II and achieves **10.6×** higher search efficiency, and also demonstrates strong scalability to a complex industrial-scale core. The code repository is available at: https://github.com/GEAR-SEU/MicroEvo-ICCAD-26.

Keywords

Microarchitecture, Design Space Exploration, Large Language Model

1 Introduction

Emerging applications are increasingly driving hardware design toward workload-specific microarchitectures [1–3]. Under this trend, designers need to search over a broad space of feasible microarchitecture configurations to identify desirable performance, power, and area (PPA) trade-offs. However, the design space of modern computer architecture is typically high-dimensional and discrete, making design space exploration (DSE) particularly challenging. The impact of microarchitecture parameters on PPA is highly nonlinear, and even small parameter changes can alter the underlying dataflow logic [4]. Moreover, PPA evaluation often has a long turnaround time, preventing timely feedback to adjust optimization decisions [5]. As a result, under the tight schedules of practical processor development, scarce evaluation opportunities must be used as efficiently as possible to approach the Pareto front.

To address this expensive multi-objective optimization (MOO) problem, industrial practice often relies on the prior knowledge of architects to directly propose candidate designs [6], but such approaches are inevitably constrained by individual bias and limited expertise. Earlier methods attempt to accelerate PPA estimation using analytical models [7, 8]. However, they require substantial modeling effort and suffer from poor transferability as microarchitectures evolve. Recent studies propose machine learning-based surrogate models to predict tool evaluations, reducing both modeling difficulty and evaluation cost [9–13]. Although these methods have significantly improved search efficiency, the limited accuracy of absolute PPA prediction in the small-sample regime of microarchitecture DSE still introduces deviations from ground truth, leading to suboptimal results [14]. Consequently, existing methods remain insufficient for the sampling-efficiency and Pareto-coverage requirements of modern microarchitecture DSE.

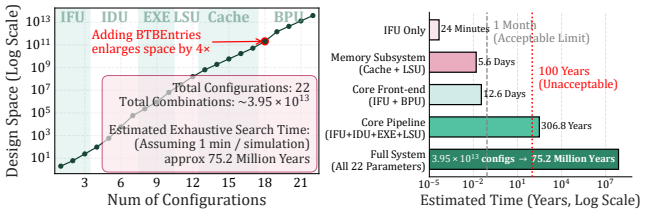
These limitations suggest that, in expensive microarchitecture DSE, the key challenge is *not only better PPA prediction, but also better sampling*. When only a small number of design evaluations can be afforded, effective DSE must **make each sampling count** by proposing informative and high-quality candidates. Lacking human prior knowledge, conventional MOO strategies (e.g., evolutionary algorithms and Bayesian optimization) rely on probabilistic sampling for blind search. Consequently, they often waste the simulation budget on inefficient configurations [15]. Large language models (LLMs) are pretrained on massive corpora containing textbooks, design documents, and other forms of microarchitecture knowledge [16]. This factor equips them with the potential to generate designs that inherently respect architectural constraints and dependencies among microarchitecture components, thereby improving sample quality. To apply an LLM in microarchitecture DSE, existing methods iteratively feed PPA feedback to the LLM to guide exploration [16–19]. However, these studies merely learn superficial numerical information from the PPA data. They fail to fully exploit the knowledge-driven generation and reasoning capabilities of the LLM to provide informative guidance for each iteration [20–23]. Consequently, the LLM acts more as a black-box numerical optimizer, thereby undermining the reliability of search decisions.

Existing data-driven DSE methods often exhibit poor sample efficiency, leading to weak optimization performance when few evaluations are affordable. To address this issue, we propose Evolution of Microarchitecture (**MicroEvo**), a knowledge-guided framework for microarchitecture DSE. The key idea is to combine the reasoning ability of LLMs with the structured search behavior of Monte Carlo Tree Search (MCTS), enabling MicroEvo to continuously learn optimization experience from DSE and evolve its search strategy. Instead of treating the LLM as a one-shot sampler or a black-box predictor, MicroEvo places it in an evolutionary search loop and equips it with a Pareto-aware tree policy, optimization knowledge, and state-aware search directives. This design transforms blind

[†]Equal Contribution.
[*]Corresponding Author: xi.wang@seu.edu.cn

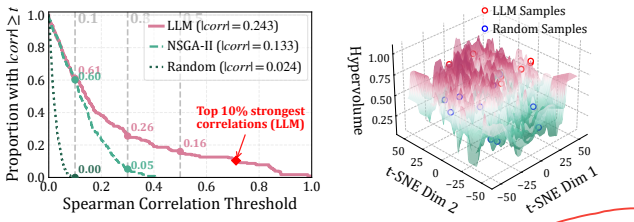


This work is licensed under a Creative Commons Attribution 4.0 International License.
ICCAD '26, San Jose, CA, USA
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2873-0/2026/11
https://doi.org/10.1145/3831252.3834100



(a) (b)

Figure 1: (a) The combinatorial explosion of the microarchitecture configurations. (b) Exhaustive search time for CPU design space.



(a) (b)

Figure 2: (a) The proportion of configuration pairs (e.g. `FetchWidth` and `RobEntries`) whose correlation strength exceeds a given threshold. A curve with a lower slope indicates a stronger correlation. (b) t-SNE visualization of microarchitecture design space.

probabilistic search into informed sampling that respects microarchitectural dependencies. Accordingly, MicroEvo can better balance exploration and exploitation and more effectively approach the Pareto front in the expensive microarchitecture DSE.

Our main contributions are summarized as follows:

- We propose an **LLM-driven evolutionary microarchitecture DSE framework** that integrates MCTS with *knowledge-guided operators*. This framework achieves informed local refinement and global design refactoring for sample-efficient exploration.
- We propose **Pareto-UCT**, an evolutionary decision tree policy for multi-objective microarchitecture optimization. It selects expansion nodes according to both Pareto quality and coverage, making MCTS more suitable for global PPA-oriented DSE.
- We introduce **Active Knowledge Accumulation (AKA)**, which extracts reusable optimization knowledge from explored designs through *Pareto analysis* and parent-child *pairwise analysis*. AKA maintains a utility-aware memory to manage and retrieve useful insights for subsequent expansions.
- We design **State-Aware Directive (SAD)** that monitors real-time DSE progress and dynamically switches among *exploit*, *balanced*, and *explore* modes. This design allows the LLM operators to adapt their modification behavior to the current search state, helping the search alleviate stagnation and local optima.
- Extensive experiments show that MicroEvo consistently achieves better Pareto quality under the same evaluation, and remains effective even in low-budget settings, with up to **36.2%** improvement in hypervolume and **10.6×** search efficiency over NSGA-II.

2 Background & Motivation

2.1 Microarchitecture DSE

Microarchitecture DSE is constrained by an enormous combinatorial space. A modern out-of-order processor exposes many interacting parameters across the front-end, execution engine, and memory hierarchy. Even with only 22 basic configurations, the feasible design space reaches approximately 3.95×10^{13} combinations, as illustrated in Figure 1a. Moreover, obtaining reliable PPA metrics typically requires cycle-accurate simulation and power modeling, making each evaluation expensive [24]. As shown in Figure 1b, even under the optimistic assumption of one minute per simulation, exhaustively searching the entire design space would require approximately 75.2 million years. This exhaustive-search estimate implies that, under realistic project schedule constraints, the available evaluation budget is extremely scarce and every simulation opportunity is precious.

This challenge is further amplified by the ruggedness of the design space, as there is complex coupling between microarchitecture configurations. For example, enlarging the ROB capacity may yield performance gains only when the fetch width and the number of functional units are matched. Otherwise, it merely increases power and area overhead. Such coupling effects make the design space highly rugged, densely populated with local optima, where even minor perturbations in parameter configurations can cause unpredictable fluctuations in PPA outcomes (Figure 2b). To visualize this ruggedness, we reduce the configurations to a 2D space using t-SNE [4], with the PPA metrics and hypervolume [25] evaluated using the experimental setup detailed in Section 4.1. As a result, unguided blind search is prone to getting trapped in local optima or wasting evaluations in low-quality regions without learning how microarchitectural components should be coordinated.

2.2 Challenges & Motivations

Given that coordination between microarchitecture configurations can introduce vast suboptimal solutions, blindly exploring the design space without professional knowledge is unwise. Incorporating prior knowledge into DSE has been recognized as a viable approach to improve exploration efficiency. BOOM-Explorer [9] leverages active learning guided by the clustering features of configurations to accelerate exploration. RL-DSE [26] models microarchitecture scaling graphs to inform configuration correlations, training an agent for component-wise parameter selection.

Challenge 1: Lack of explicit knowledge guidance. While these methods leverage prior knowledge to accelerate exploration, such experience is implicitly learned through task-specific models. This task-specific experience incurs high portability costs across architectures and cannot be directly queried or reused during DSE.

Why LLM. The successful application of LLMs to chip design and verification has inspired their adoption in hardware DSE [27–33]. As shown in Figure 2a, LLM-generated samples (DeepSeek-V3.2) exhibit stronger parameter correlations. This correlation pattern suggests that pretrained knowledge enables the LLM to capture the intrinsic configuration coupling. Moreover, when these samples are mapped into the objective space (Figure 2b), LLM-generated points identify high-quality Pareto regions (distributed in the upper half-space along the z-axis) compared to random sampling. These results highlight LLMs as promising knowledge-driven samplers. Recent LLM-driven DSE methods have also shown strong potential [16, 17]. As Figure 3 (left) shows, they quickly approach a broad Pareto front, as reflected by the steep rise of the hypervolume curve.

Challenge 2: Ineffective utilization of iterative feedback. Despite their high-quality sampling, existing LLM-driven DSE methods struggle to translate the knowledge advantages of LLMs into sustained optimization benefits. As exploration progresses, the lack of high-value iterative feedback to inform further search decisions often leads to a plateau in multi-objective optimization efficiency.

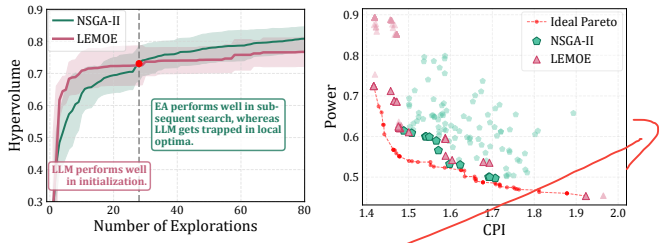


Figure 3: Left: Comparison of the LLM-driven and EA-based DSE methods. Right: Search results in the CPI–Power space.

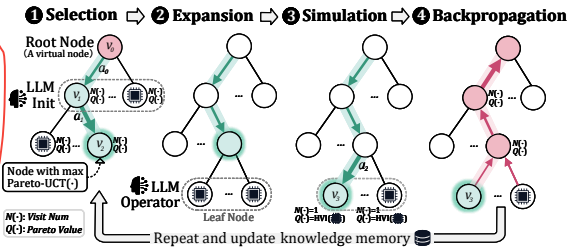


Figure 4: Evolutionary Monte Carlo Tree Search (MCTS) procedure.

Why Evolutionary Iteration. Without effective search state management and diversity preservation mechanisms, LLM-driven sampling tends to repeatedly exploit a few successful design patterns, leading to premature convergence in local optima. In contrast, as shown in Figure 3, traditional evolutionary algorithms maintain a stable optimization trajectory through population control, even though they start slower due to blind probabilistic operators. Therefore, combining the knowledge-guided generation of LLMs with the evolutionary framework offers a promising direction for high-efficiency sampling, sustained exploration, and global convergence.

3 Design and Methodology

This section presents the **MicroEvo** framework (Figure 5). It formulates microarchitecture DSE as an evolutionary search process organized by Monte Carlo Tree Search (MCTS). The LLM first initializes a set of root designs targeting different optimization preferences. After selecting promising nodes using a multi-objective UCT criterion, MicroEvo applies LLM-driven evolutionary operators to expand the search tree, while learning from accumulated exploration knowledge and dynamically adapting the expansion directive. Each expanded node corresponds to a newly refined microarchitecture design, further steering the search toward the Pareto front.

3.1 Evolutionary Decision Tree Search

We instantiate MCTS as the search backbone of MicroEvo, where each tree expansion is implemented by LLM-driven mutation and crossover operators. In this formulation, MCTS provides structured selection and backpropagation [34, 35], while the LLM serves as the generator of new designs. In the search tree, each node (*state* v_i) corresponds to a microarchitecture design with specific parameters. Each edge in Figure 4 represents an *action* a_i that modifies the current design, such as tuning configurations. The PPA simulation results of each optimized design serve as the *reward function* $Q(v_i)$ for *state* v_i. At each iteration, MCTS performs four stages:

❶ *Selection.* Selection balances exploration and exploitation in MCTS. Starting from the root node, MicroEvo recursively selects the child with the highest upper confidence bound until reaching a leaf node. A standard UCT criterion is defined as:

$$\mathrm{UCT}(v_i) = Q(v_i) + \lambda\sqrt{\frac{\ln N(v_p)}{N(v_i)}} \quad (1)$$

where $Q(v_i)$ denotes the reward of child node v_i, $N(v_i)$ is its visit count, v_p is its parent node, and λ is the exploration coefficient. Unlike elite selection in genetic algorithms, UCT trades off current value and visit frequency, thereby allowing the search to revisit sub-optimal yet potentially promising nodes. In our design, we further improve the UCT using multi-objective metrics (Section 3.2).

❷ *Expansion.* Given the selected leaf node, MicroEvo invokes LLM-driven evolutionary operators to generate new candidate designs. Rather than relying on hand-crafted heuristics, these operators propose knowledge-guided modifications based on the current search state and historical exploration (Section 3.3).

❸ *Simulation.* In the simulation stage, the expanded design is evaluated using the target toolchain to obtain its PPA metrics.

❹ *Backpropagation.* The evaluation result is propagated along the path to update values and visit counts of all encountered nodes.

After a predefined simulation count (budget) is exhausted, the MCTS tree contains a set of high-quality microarchitecture designs.

3.2 Pareto-UCT

Selecting which nodes to expand is a crucial step in MCTS [34]. In Section 3.1, the standard UCT rule is defined by a single-objective reward. However, multi-objective microarchitecture DSE requires balancing competing PPA objectives, where search quality should be judged by its contribution to the global Pareto front rather than a single scalar. To address this mismatch, we propose **Pareto-UCT**, a selection criterion for global search. For a child node v_i, its Pareto-UCT score is defined as:

$$\text{Pareto-UCT}(v_i) = Q_{\mathrm{HVI}}(v_i) + e\,D(v_i) + \lambda\sqrt{\frac{\ln N(v_p)}{N(v_i)}} \quad (2)$$

where v_p denotes the parent node of v_i, $N(v_p)$ and $N(v_i)$ are the visit counts of the parent and child nodes, respectively.

The first term $Q_{\mathrm{HVI}}(v_i)$ in Eq. (2) quantifies the contribution of the current branch to the global Pareto front, measured by the hypervolume improvement (HVI) of node v_i:

$$\mathrm{HV}(\mathcal{P}) = \Lambda\left(\bigcup_{i=1}^{|\mathcal{P}|}[\boldsymbol{f}(v_i), \boldsymbol{r}]\right) \quad (3)$$

$$Q_{\mathrm{HVI}}(v_i) = \mathrm{HV}\big(\mathcal{P} \cup \{\boldsymbol{f}(v_i)\}\big) - \mathrm{HV}(\mathcal{P}) \quad (4)$$

where $\mathcal{P}$ denotes the current non-dominated set, $\Lambda(\cdot)$ represents the hyper-rectangle bounded by the objective vector $\boldsymbol{f}$ and the reference point $\boldsymbol{r}$. Intuitively, $Q_{\mathrm{HVI}}(v_i)$ measures the marginal contribution of v_i to expanding the dominated objective space volume.

The second term, $D(v_i)$ weighted by e, represents the crowding distance in objective space. This term prioritizes sparsely sampled regions, encouraging the search to cover under-explored parts of the Pareto front. Let C be the set of valid simulated children of v_p. We form a comparison set $\mathcal{S} = \mathcal{P} \cup C$. For each node $v_i \in \mathcal{S}$, $D(v_i)$ accumulates the normalized distance between adjacent neighbors across all objective dimensions $\mathcal{M}$:

$$D(v_i) = \sum_{m=1}^{\mathcal{M}} \frac{f_m(v_i^{+,m}) - f_m(v_i^{-,m})}{f_m^{\max} - f_m^{\min}} \quad (5)$$

where $\mathcal{M} = 3$ corresponds to the PPA objectives, and $f_m^{\max}$ and $f_m^{\min}$ are the maximum and minimum values of objective m in $\mathcal{S}$.

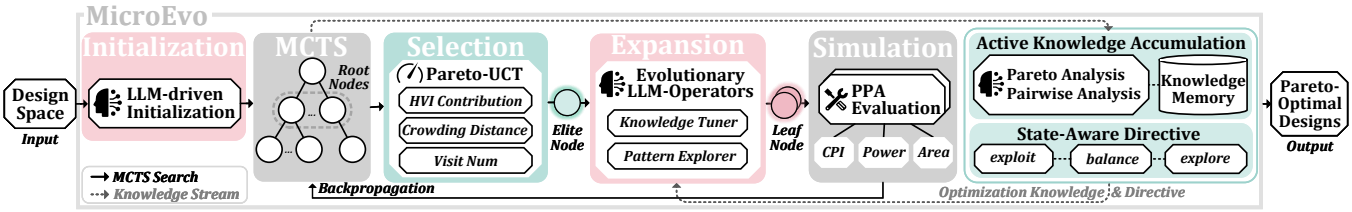


Figure 5: Overview of MicroEvo. LLM first initializes root designs, Pareto-UCT then selects elite nodes in the MCTS tree *(Section 3.2)*, LLM-driven operators expand new nodes *(Section 3.3)*, and PPA evaluation updates the tree, while active knowledge accumulation *(Section 3.4)* and state-aware directives *(Section 3.5)* continuously provide optimization guidance until the simulation budget is reached.

The terms $v_i^{+,m}$ and $v_i^{-,m}$ denote the immediate upper and lower neighbors of v_i when the set $\mathcal{S}$ is sorted along objective m.

The third term ensures that infrequently visited and suboptimal yet promising nodes still have a chance to be explored:

$$\lambda = \lambda_0 \sqrt{\frac{T-t}{T}} \tag{6}$$

where λ is a dynamically decayed exploration coefficient, with λ_0 as the initial weight, t the current iteration, and T the total search budget. This schedule gradually shifts the search from early exploration to later exploitation.

Using this criterion, MicroEvo selects an elite leaf node for subsequent expansion. Compared with standard UCT, Pareto-UCT preserves the exploration–exploitation trade-off while incorporating Pareto front quality and distribution diversity, making it more suitable for multi-objective microarchitecture DSE.

3.3 LLM-Driven Initialization & Expansion

Unlike traditional MCTS that builds from a single node, MicroEvo begins from a virtual root where the LLM generates an initial set of microarchitecture designs. To provide a broader spectrum of search directions early in DSE, we instruct the LLM to generate distinct parameter combinations targeting specific PPA trade-offs: maximum performance, maximum efficiency, balanced, performance-leaning, and power/area-leaning. The LLM selects configurations from a user-defined design space, leveraging its internalized knowledge of microarchitectural correlations to produce effective designs. To handle potential LLM hallucinations, any generated parameter outside the design space is mapped to the nearest valid value.

Once the initial set is evaluated and the MCTS tree is established, Pareto-UCT continuously selects elite nodes for expansion. To better inform subsequent search decisions, MicroEvo integrates PPA feedback and expert knowledge into structured prompts that drive the LLM to propose new children, as shown in Figure 6. Inspired by the mutation and crossover mechanisms in evolutionary algorithms [36, 37], we design two distinct LLM-driven expansion operators: the **knowledge tuner** and the **pattern explorer**.

The *knowledge tuner* leverages global insights (Section 3.4) and parent node PPA results for local improvements. Based on the accumulated optimization knowledge, it avoids inefficient combinations. For example, if historical experience indicates that scaling the reorder buffer (ROB) requires increasing load/store queues to prevent stalls, the LLM adjusts these coupled parameters accordingly. Moreover, by identifying objectives lagging behind the Pareto front (e.g., power efficiency), the LLM translates this gap into targeted refinements, such as downsizing caches or issue widths.

The *pattern explorer* goes beyond random parameter mutation to identify higher-level optimization patterns. Given sibling nodes that share the same parent, the LLM analyzes their configurations and PPAs to propose novel solutions relative to the current set of explored nodes. For example, recognizing a "high fetch/issue width + large cache capacity" pattern suited for complex workloads, it may propose an alternative optimization regime. In this way, the mutation operates at the level of architectural design intent rather than superficial parameter perturbation.

Algorithm 1 Active Knowledge Accumulation with Knowledge-Guided LLM Expansion

Input: selected node v, sibling set $\mathcal{S}(v)$, Pareto set $\mathcal{P}$, insight memory Ω
Output: child node C_{new}, updated Pareto set $\mathcal{P}^*$, updated memory Ω^*

report $\leftarrow$ ParetoAnalysis$(v, \mathcal{P})$ ▷ returns Pareto knowledge
for all $k \in \Omega$ **do**
  compute utility $U(k) = \widetilde{Q}_{\text{HVI}}(k) - \omega \log(N(k)+1)$
end for
$\mathcal{K} \leftarrow$ Top-K(Ω, U) ▷ retrieve the most useful insights
$C_r \leftarrow$ KnowledgeTuner$(v, \textit{report}, \mathcal{K})$ ▷ guided by Pareto and pairwise analysis
$C_p \leftarrow$ PatternExplorer$(v, \mathcal{S}(v))$ ▷ exploration using sibling context
$C_{\text{new}} \leftarrow C_r \cup C_p$
for all $c \in C_{\text{new}}$ **do**
  CPI, Power, Area $\leftarrow$ Evaluate(c)
  HVI$(c) \leftarrow$ NonDominatedSort$(c, \mathcal{P})$
  if HVI$(c) > 0.01$ **or** $\exists j,\ (f_j(v) - f_j(c))/f_j(v) > 20\%$ **then**
    $k_{\text{new}} \leftarrow$ PairwiseAnalysis(v, c) ▷ extract from parent-child PPA change
    if NotDuplicate(k_{new}, Ω) **then** ▷ similarity check via embedding
      $\Omega^* \leftarrow \Omega \cup \{k_{\text{new}}\}$
    end if
  end if
end for
update $N(k)$, Pareto-optimal design set, and cumulative HVI gain
return $C_{\text{new}}, \mathcal{P}^*, \Omega^*$

3.4 Active Knowledge Accumulation

To extract practical experience and inform LLM decision-making during expansion, we introduce an **active knowledge accumulation (AKA)** mechanism together with a *knowledge memory* to manage experience extraction and application. Whenever a child node is expanded, the LLM reflects on the optimization trajectory and summarizes reusable insights (Algorithm 1). These generated insights are grounded in the PPA changes and are directly relevant to the ongoing DSE process. Therefore, AKA transforms process-level experiences into reusable knowledge across two levels.

Pareto Analysis. By performing non-dominated sorting over the explored designs, AKA identifies the current Pareto-optimal microarchitectures and analyzes the coverage of the explored front. This analysis allows the LLM to summarize multi-objective optimization insights across diverse design preferences. It learns to construct parameter combinations that approximate the broad Pareto front and extracts global optimization patterns. Moreover, it diagnoses which regions of the objective space remain under-explored and suggests the next search direction. To maintain solution diversity, this guidance avoids prescribing exact configuration edits.

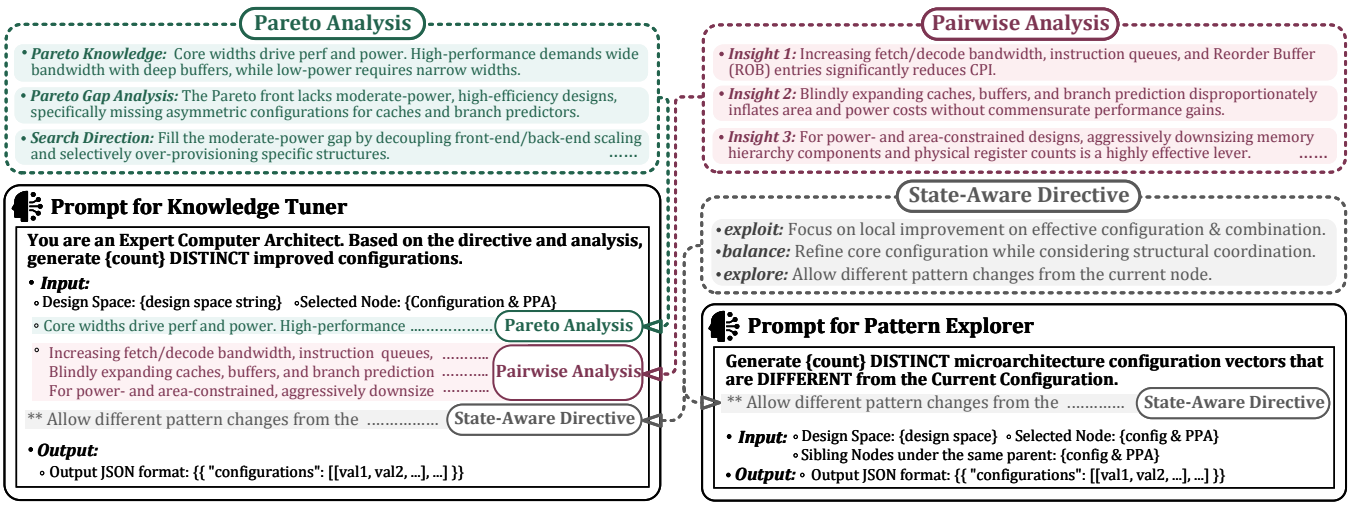


Figure 6: Prompt integration for LLM-driven expansion.

Pairwise Analysis. AKA compares each expanded child with its parent, correlating specific parameter adjustments with resulting PPA variations to translate raw simulation feedback into actionable tuning rules. For instance, if reducing L2 cache associativity decreases power consumption with minimal performance loss, the LLM formulates a guideline prioritizing this for power-constrained designs. This process extracts fine-grained procedural experience on beneficial configuration updates. To avoid accumulating trivial knowledge, extraction is triggered only when the child delivers a significant HVI gain or clear improvement in at least one objective. To ensure that the extracted insights are robust to potential absolute simulation errors from analytical tools (e.g., McPAT), the LLM is instructed to avoid generating specific quantitative mappings.

Since not all extracted insights are equally valuable, AKA employs a **knowledge memory** with redundancy-aware insertion and utility-aware retrieval. Each memory item stores one reusable insight with its historical application records. New insights are semantically filtered before insertion to remove redundant items. When insight k is used, AKA records the expanded child node v_t and the obtained HVI. The memory module then scores each stored insight and retrieves the highest-scoring items to guide the next expansion. The utility score U of insight k is defined as:

$$U(k) = \underbrace{\tilde{Q}_{\text{HVI}}(v_i \mid k)}_{\textit{Pareto Reward}} - \underbrace{\omega \log(N(k)+1)}_{\textit{Visit Penalty}} \tag{7}$$

$$\tilde{Q}_{\text{HVI}}(v_i \mid k) = \frac{Q_{\text{HVI}}(v_i \mid k) - Q^{min}_{\text{HVI}}(v_i)}{Q^{max}_{\text{HVI}}(v_i) - Q^{min}_{\text{HVI}}(v_i) + \epsilon} \tag{8}$$

where $N(k)$ is the usage count of k, ω controls the penalty strength, and ϵ is a small constant to prevent division by zero. The utility score balances two factors: *Pareto Reward* uses normalized HVI gain to reflect the contribution to the Pareto front. *Visit Penalty* suppresses frequently reused insights to encourage visiting newer knowledge. This memory mechanism prevents a few rules from dominating the search while prioritizing consistently effective and recently successful insights, ensuring that the retrieved knowledge remains reusable and adaptive as the search trajectory evolves.

3.5 State-Aware Directive

A fixed prompting strategy is often insufficient for LLM-driven microarchitecture DSE [18]. When the search progresses well, the framework should favor minor refinements around promising designs. In contrast, when the search stalls, repeated local modifications waste the simulation budget. To address this issue, we introduce a **state-aware directive (SAD)** mechanism, which monitors the search progress and dynamically adjusts the directive given to the LLM during node expansion (Figure 7).

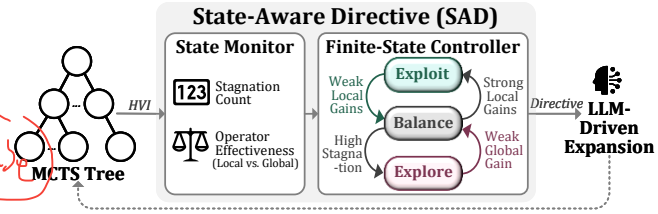


Figure 7: State-Aware Directive (SAD) workflow

SAD operates as a finite-state controller with three modes: *exploit*, *balance*, and *explore*. It tracks two runtime signals. The first is `stagnation_count`, which records the number of consecutive evaluations without meaningful HVI (1% of the total HV) and resets upon any significant HVI gain. The second is the recent effectiveness of expansion operators, indicating whether the current search is benefiting more from local refinement or from broader structural changes. Based on these signals, SAD updates the search mode in real time. If `stagnation_count` exceeds a predefined threshold, the search is considered trapped locally, and the controller switches to *explore*. If stagnation is low and recent local refinements consistently improve the Pareto front, it switches to *exploit*. In all other cases, the controller stays in *balance*.

The selected mode provides natural language directives to the LLM operator for expansion. In *exploit* mode, SAD injects a directive to the prompt to make local adjustments, preserving the main combination pattern. In *balance* mode, the LLM is guided to introduce moderate changes that consider configuration dependencies. In *explore* mode, the directive discourages minor edits and encourages larger structural changes targeting under-explored Pareto regions. When the current branch shows limited recent progress, this strategy helps the search move away from local design patterns. By introducing SAD, MicroEvo avoids relying on a static expansion policy. Instead, it dynamically adapts LLM-generated modifications

Table 1: Comparison of different DSE methods under two search budgets. MicroEvo is evaluated with DeepSeek-V3.2 and Gemini-3-pro.

Method	Evaluation Budget = 20						Evaluation Budget = 45					
	HV Mean ↑ ± Std	Imp. ↑	ADRS Mean ↓ ± Std	Imp. ↑	IPC/Power ↑	IPC/Area ↑	HV Mean ↑ ± Std	Imp. ↑	ADRS Mean ↓ ± Std	Imp. ↑	IPC/Power ↑	IPC/Area ↑
NSGA-II	0.5826 ± 0.0439	0.00%	0.1666 ± 0.0213	0.00%	1.1823	0.0487	0.663 ± 0.022	0.00%	0.123 ± 0.017	0.00%	1.1823	0.0524
MOTPE [38]	0.6058 ± 0.0261	+3.99%	0.1454 ± 0.0171	+12.76%	1.1762	0.0510	0.698 ± 0.026	+5.42%	0.100 ± 0.011	+18.95%	1.2053	0.0513
LEMOE [16]	0.6746 ± 0.0268	+15.80%	0.1315 ± 0.0249	+21.11%	1.1568	0.0498	0.679 ± 0.024	+2.53%	0.128 ± 0.021	-3.87%	1.1568	0.0498
Boom-Explorer [9]	0.6203 ± 0.0528	+6.48%	0.1562 ± 0.0194	+6.29%	1.1854	0.0492	0.709 ± 0.042	+6.97%	0.100 ± 0.010	+19.12%	1.2276	**0.0537**
RL-DSE [26]	0.6340 ± 0.0577	+8.82%	0.1444 ± 0.0259	+13.33%	1.1604	0.0526	0.711 ± 0.012	+7.36%	0.105 ± 0.008	+14.46%	1.1942	0.0526
MicroEvo (DeepSeek)	0.7822 ± 0.0179	+34.27%	0.0772 ± 0.0112	+53.65%	**1.2156**	0.0521	0.817 ± 0.007	+23.35%	0.056 ± 0.005	+54.68%	1.2156	0.0523
MicroEvo (Gemini)	**0.7936 ± 0.0059**	**+36.22%**	**0.0716 ± 0.0063**	**+57.06%**	1.1951	**0.0527**	**0.821 ± 0.005**	**+23.94%**	**0.056 ± 0.003**	**+54.80%**	**1.2360**	0.0530

* **Bold** and underline indicate the best and second-best results, respectively. HV Imp. is calculated as the relative improvement compared to the baseline DSE method, whereas ADRS is the relative reduction.

Table 2: Microarchitecture Design Space Configuration

Components	Candidate Values	Components	Candidate Values
Alpha21264		**XiangShan (continued)**	
FetchWidth	4, 8	phyregReleaseWidthInt	1, 2, 4, 6, 8
FetchBufferSize	16, 32, 48, 64	phyregReleaseWidthFp	1, 2, 4, 6, 8
FetchQueueSize	16, 32, 48, 64	phyregReleaseWidthVec	1, 2, 4
DecodeWidth	2, 4, 6, 8	SXQSize	12:36:6
ROBSize	32:192:32	VXQSize	12:36:6
IntPhyRegister	32:320:32	MXQSize	12:36:6
FpPhyRegister	32:320:32	SXQEnqSize	1, 2, 3
DispatchWidth	2, 4, 6, 8	VXQEnqSize	1, 2, 3
IssueWidth	2, 4, 6, 8	MXQEnqSize	1, 2, 3
IQEntries	16:96:8	LSQSize	12:36:6
LDQEntries	16, 24, 32, 40, 48	LSQEnqSize	1, 2, 3
STQEntries	16, 24, 32, 40, 48	LoadRequestQueueSize	16, 32, 64
TLBSize	64, 96, 128, 160	TempStoreQueueSize	16, 32, 64
ICacheSize	16 kB, 32 kB, 64 kB	StoreMergeBuffer	8, 16, 32
ICacheAssociativity	2, 4, 8	RARQSize	24:128:8
DCacheSize	16 kB, 32 kB, 64 kB	RAWQSize	24:96:8
DCacheAssociativity	2, 4, 8	LFSTSize	8, 16, 32
BTBEntries	1024, 2048, 4096, 8192	LFSTEntrySize	1, 2, 3, 4
RASSize	16:40:4	SSITSize	8, 16, 32
LocalPredictorSize	512, 1024, 2048	cacheLoadPorts	1:8:1
GlobalPredictorSize	2048, 4096, 8192	cacheStorePorts	1:8:1
ChoicePredictorSize	2048, 4096, 8192	TLBSize	32, 64, 128, 256
Design Space: $\approx 3.95 \times 10^{13}$		ICacheSize	16 kB, 32 kB, 64 kB
XiangShan		ICacheAssociativity	4, 8
FetchWidth	4, 8, 16	L1DcacheSize	16 kB, 32 kB, 64 kB
FetchQueueSize	16:96:16	DCacheAssociativity	4, 8
DecodeWidth	4:12:2	MSHRSize	8, 16, 32
RenameWidth	4:12:2	uBTBSize	512, 1024, 2048
ROBSize	160:640:40	L1BTBSize	2048, 4096, 8192
IntPregsNum	160:640:20	L2BTBSize	4096, 8192, 16384, 32768
FloatPregsNum	96:316:20	RASSize	16, 32, 64
VectorPregsNum	48:96:8	**Design Space:** $\approx 3.41 \times 10^{26}$	
PredicateVectorPregsNum	96:368:16		

according to the observed search state, allowing DSE to efficiently alternate between local refinement and broader exploration.

4 Evaluations

4.1 Experimental Setup

Design Space. We employ GEM5 [39] for cycle-accurate simulation and McPAT [40] for power and area modeling. The primary evaluated processor is an out-of-order RISC-V core with a two-level cache hierarchy and 16 GB DRAM, with its design space detailed in Table 2. For each design point, the PPA metrics are reported as the geometric mean across the RISC-V benchmark suite [41]. To further validate the generalization and transferability of our framework, we also evaluate the XiangShan Kunminghu processor [42] via XS-GEM5 [43]. This processor is substantially more complex, featuring a TAGE branch predictor and multiple hardware prefetchers.

Baselines. We compare our method with NSGA-II, MOTPE [38], Boom-Explorer (Bayesian optimization) [9], RL-DSE (reinforcement learning) [26], and LEMOE (LLM-driven) [16]. For RL-DSE, we allocate the evaluation budgets to obtain uniformly sampled preference vectors to learn the convex coverage set. Among prior LLM-driven DSE methods [18, 19], we select LEMOE as the most representative recently published approach for this problem and reproduce it with DeepSeek-V3.2. All experiments were conducted on an Intel Xeon Platinum 8480+ server and averaged over five independent trials.

DSE Performance. We compare these DSE methods using hypervolume (HV) and average distance to reference set (ADRS). For HV, the reference point is defined by taking the maximum value along each objective. For ADRS, we construct the reference Pareto front $\mathcal{P}^*$ by combining 5,000 randomly sampled design points with all solutions explored by the evaluated methods:

$$\mathrm{ADRS}(\mathcal{P}, \mathcal{P}^*) = \frac{1}{|P^*|} \sum_{x^* \in \mathcal{P}^*} \min_{x \in \mathcal{P}} \delta(x, x^*) \tag{9}$$

$$\delta(x, x^*) = \max_{m \in \mathcal{M}} \left\{ 0, \frac{f_m(x) - f_m(x^*)}{f_m(x^*)} \right\} \tag{10}$$

Here, $\mathcal{M}$ denotes the set of objectives. A lower ADRS indicates that the explored Pareto set is closer to the reference front.

To evaluate sampling efficiency, we measure the number of evaluations t required for each method to reach a target threshold τ:

$$N_\tau = \min\{t \mid \mathrm{HV}(t) \geq \tau\}, \tag{11}$$

where $\mathrm{HV}(t)$ denotes the best HV achieved by a DSE method after t evaluations. A smaller N_τ indicates higher search efficiency.

Hyperparameter Settings. The MicroEvo process runs for 10 iterations, starting from five initially sampled points (for a total of 45 simulations). During each expansion step, the two LLM operators each generate two new candidate designs. At each iteration, three insights are injected into the search. Newly generated insights are encoded via MiniLM embeddings, and semantic redundancies are filtered using a similarity threshold of 0.75. For Pareto-UCT, the parameter e is set to 0.05 and λ_0 is set to 1.414. In Active Knowledge Accumulation (AKA), the penalty coefficient for repeated access to the same insight is 0.3.

4.2 LLM-Driven MicroEvo Versus Prior Arts

Pareto quality comparison. Table 1 summarizes the optimization quality of the evaluated DSE methods. MicroEvo with Gemini-3-pro improves HV by 23.9% over NSGA-II, 17.0% over Boom-Explorer, 16.6% over RL-DSE, and 21.4% over LEMOE. Figure 8 further shows that MicroEvo achieves better Pareto front coverage and stays closer to the reference front under the same evaluation count. When the budget is reduced to only 20 evaluations (4 iterations for MicroEvo), it still identifies a broader set of Pareto-optimal solutions. This result is important for practical processor development, where the available simulation budget is typically very limited.

Traditional multi-objective optimization algorithms, such as NSGA-II and MOTPE, lack architectural domain knowledge and struggle to converge rapidly in a highly rugged design space. While Boom-Explorer benefits from Bayesian optimization (BO), its later search quality is constrained by Gaussian process prediction errors, leading to suboptimal results. RL-DSE requires iterative policy

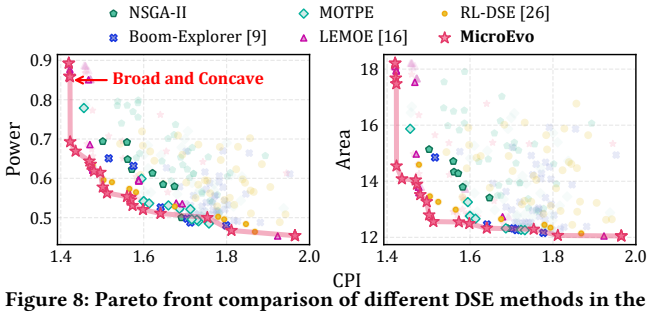


Figure 8: Pareto front comparison of different DSE methods in the CPI–Power and CPI–Area spaces.

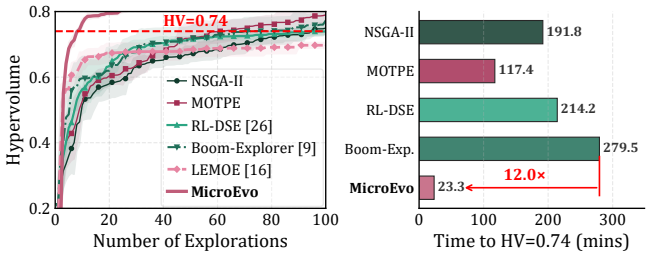


Figure 9: Convergence efficiency and time-to-target comparison.

training before it can provide useful actions, making DSE inefficient under tight budgets. Furthermore, its learned knowledge is implicitly encoded in model parameters during training, rather than explicitly extracted and reused to guide exploration. LEMOE uses the same backbone LLM as MicroEvo, but still exhibits a clear performance gap. As illustrated in Figure 10, this gap primarily emerges during the later search stages. Although LEMOE follows a BO-style iterative process, it makes limited use of PPA metrics to inform subsequent decisions. Moreover, the bias introduced by LLM-based PPA predictions further degrades candidate selection [19]. In contrast, MicroEvo continuously accumulates useful microarchitecture insights via AKA, while State-Aware Directive (SAD) dynamically adjusts search directives based on exploration status. Consequently, MicroEvo achieves a balance between exploration and exploitation.

Comparison of sample efficiency. To further evaluate search efficiency, we record the number of evaluations required to reach the target HV threshold ($HV = 0.74$) in Figure 9. The results show that MicroEvo (Gemini) requires significantly fewer evaluations than the baselines, improving search efficiency by **10.6×** compared to NSGA-II, and consumes only **8.3%** of Boom-Explorer's execution time to reach this target. In the early DSE stage, the synergy of Pareto analysis and LLM-pretrained knowledge of microarchitectural component interdependencies enables MicroEvo to rapidly approximate the Pareto front. This synergy ensures that each trial maintains structural validity against configuration coordinations and preserves a global view of the design space. In the later stage, accumulated knowledge and SAD allow the framework to dynamically adapt search directives based on the current state, helping the search escape local optima. Meanwhile, Pareto-UCT ensures that exploration focuses on the most promising nodes. It exploits the architectural patterns of the current best solutions and extracts valuable features from suboptimal designs, translating them into effective directions for subsequent exploration.

Practical overhead and search trajectory. To quantify the LLM overhead in MicroEvo, Figure 11 reports the response time, token consumption, and API cost of each LLM-driven stage when using DeepSeek-V3.2. In one complete MicroEvo search, including

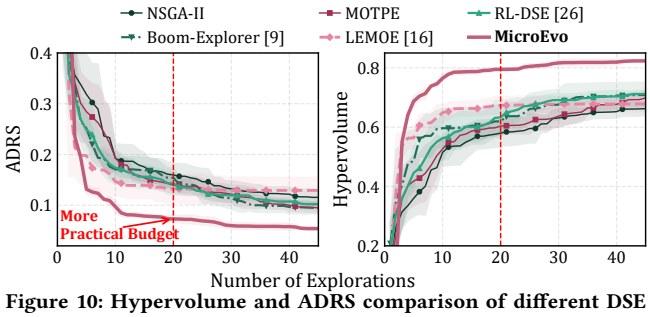
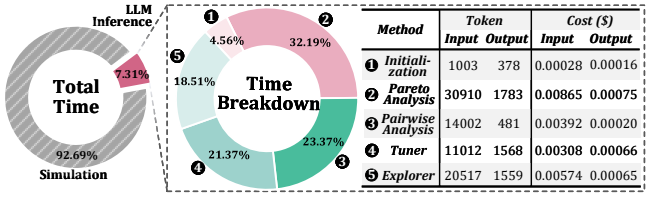
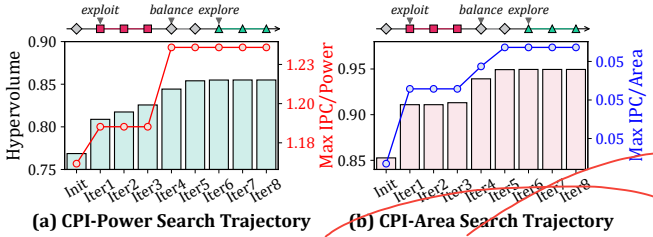


Figure 10: Hypervolume and ADRS comparison of different DSE methods versus number of explorations: the red dashed line marks a more practical evaluation budget of 20 trials.



Method	Token Input	Token Output	Cost ($) Input	Cost ($) Output
❶ Initialization	1003	378	0.00028	0.00016
❷ Pareto Analysis	30910	1783	0.00865	0.00075
❸ Pairwise Analysis	14002	481	0.00392	0.00020
❹ Tuner	11012	1568	0.00308	0.00066
❺ Explorer	20517	1559	0.00574	0.00065

Figure 11: Token usage, cost, and total runtime breakdown of MicroEvo (DeepSeek-V3.2): 7.3% LLM inference vs. 92.7% simulation.



(a) CPI-Power Search Trajectory **(b) CPI-Area Search Trajectory**

Figure 12: MCTS Search Progress over Iterations.

initial sampling stage and ten rounds of iterative expansion, the total LLM invocation time is only 5.85 minutes, with an API cost of approximately $0.024 (34.52 minutes of LLM invocation and $1.32 API cost for Gemini-3-pro). LLM inference accounts for only 7.31% of the overall runtime, indicating that MicroEvo remains practical and scalable for microarchitecture DSE. Figure 12 shows that MicroEvo steadily improves search quality over iterations and reaches strong optimization performance with few iterations. The switching among *exploit*, *balance*, and *explore* modes further suggests that the state-aware directive can adapt the search behavior according to the current progress, enabling effective coordination between local refinement and broader exploration.

4.3 Ablation Study

Learning from iterative search. To validate the necessity of each MicroEvo component for efficient multi-objective optimization, we conduct an ablation study (Table 3). Removing both active knowledge accumulation and the search directive (*w/o AKA, SAD*) significantly degrades performance. This indicates that simply combining LLMs with tree search is not sufficient for microarchitecture DSE. Without actionable knowledge support and search-state control, the framework fails to effectively leverage historical exploration signals, and its optimization decision-making becomes much less reliable under limited evaluations. We further isolate the two LLM operators by retaining only the *pattern explorer* (*w/o Tuner*) and generating four new nodes at each step, or retaining only the *knowledge*

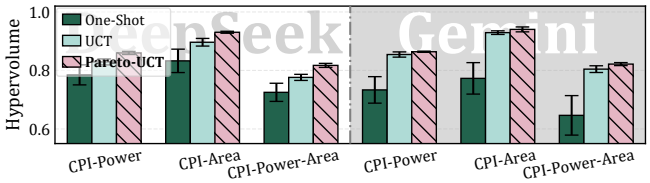


Figure 13: Effectiveness of the proposed Pareto-UCT.

Table 3: Ablation study of MicroEvo under two evaluation budgets. Improvement (Imp.) is computed with respect to the full MicroEvo.

	Evaluation Budget = 20				Evaluation Budget = 45			
	HV ↑	Imp. ↑	ADRS ↓	Imp. ↑	HV ↑	Imp. ↑	ADRS ↓	Imp. ↑
MicroEvo (Full)	**0.782**	0.0%	**0.077**	0.0%	**0.817**	0.0%	**0.060**	0.0%
w/o AKA, SAD	0.726	-7.16%	0.111	-44.73%	0.762	-6.73%	0.092	-52.49%
w/o Tuner	0.717	-8.33%	0.114	-47.78%	0.760	-6.98%	0.087	-44.60%
w/o Explorer	0.727	-6.93%	0.101	-31.71%	0.746	-8.63%	0.095	-57.81%
w/o AKA	0.728	-6.83%	0.110	-42.95%	0.771	-5.54%	0.083	-38.48%
w/o Pareto	0.746	-4.58%	0.100	-29.88%	0.786	-3.75%	0.074	-23.99%
w/o Pairwise	0.735	-6.01%	0.101	-31.86%	0.775	-5.12%	0.081	-35.39%
w/o Utility	0.737	-5.69%	0.097	-25.82%	0.784	-4.01%	0.073	-22.05%
w/o SAD	0.727	-7.03%	0.107	-38.73%	0.773	-5.34%	0.082	-35.86%

tuner (*w/o Explorer*). In both cases, the DSE performance degrades even further. The *tuner* ensures robust local refinements around promising designs, while the *explorer* introduces structural diversity to prevent overly conservative search. Their collaboration is important for sustaining search quality across iterations.

Within AKA, removing either Pareto analysis (*w/o Pareto*) or pairwise analysis (*w/o Pairwise*) yields worse results, highlighting the need for both a global optimization view and localized microarchitecture insights. The larger decrease observed in *w/o Pairwise* suggests that extracting actionable optimization knowledge from concrete improvement cases is particularly critical. Moreover, replacing utility-aware memory selection with random retrieval (*w/o Utility*) reduces performance. This shows that effective exploration relies not only on accumulating experience, but also on retrieving the most relevant insights at the right time. The utility mechanism enables flexible knowledge application, dynamically guiding microarchitecture optimization across changing search states. In the last row, we evaluate the effect of the state-aware directive. After removing the directive (*w/o Directive*), the search quality drops again, verifying the importance of dynamic search state control.

Effects of Pareto-UCT and iterative MCTS search. Figure 13 compares Pareto-UCT with a standard single-objective UCT policy. By introducing a multi-objective reward together with crowding-aware selection, Pareto-UCT more effectively identifies promising nodes within the MCTS tree. In contrast, single-objective UCT tends to over-exploit locally favorable regions and is more likely to get trapped in local optima. We also evaluate a one-shot LLM generation baseline that directly outputs 50 various design points. The MCTS-based MicroEvo achieves a clear advantage over the baseline, underscoring the importance of iterative learning in microarchitecture DSE, where effective decision-making should be continuously refined using feedback from prior evaluations. Moreover, the performance gap implies that existing LLMs cannot simply rely on memorizing optimal designs from their pre-training data.

4.4 Scaling to More Complex Microarchitecture

In terms of portability, we further evaluate MicroEvo on the XiangShan Kunminghu processor using XS-GEM5. Compared with the Alpha21264-style OoO core, XiangShan features stronger parameter coupling and a more rugged search space. Figure 14 shows

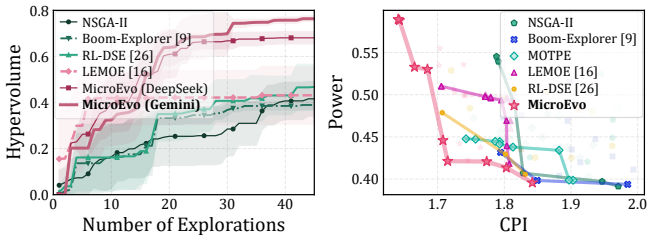


Figure 14: Left: Hypervolume trajectory during the DSE process. Right: Explored Pareto fronts in the CPI-Power objective space.

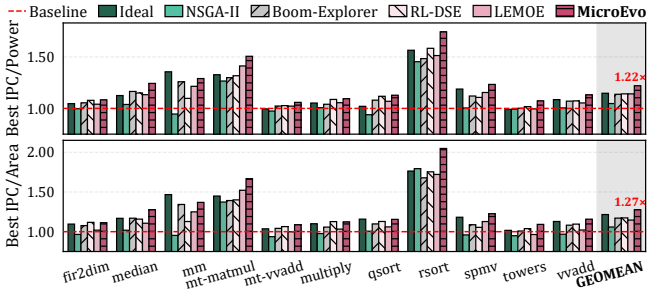


Figure 15: Relative IPC/Power and IPC/Area improvement over the RTL-aligned XiangShan Kunminghu baseline across benchmarks.

that MicroEvo still achieves the best HV trajectory and a broader, more concave Pareto front. Unguided sampling struggles to identify valuable points in vast combinatorial spaces, while surrogate-based methods are vulnerable to accumulated prediction errors. MicroEvo maintains strong sample efficiency through LLM-guided high-quality expansion and appropriate knowledge application, allowing it to scale to increasing microarchitectural complexity.

Comparison with Expert-Crafted Design. Figure 15 compares the best IPC/Power and IPC/Area explored, using the RTL-aligned Kunminghu configuration as baseline and the *Ideal* handcrafted XS-GEM5 design as an expert reference [43]. MicroEvo consistently outperforms existing DSE methods, surpassing the manual design by 8.6% in geometric mean energy efficiency. The advantage is most evident on *mm*, *rsort*, and *spmv*, where efficient coordination across the memory hierarchy and control-flow resources matters more than simply increasing peak width. This improvement comes from the ability of MicroEvo to use optimization insights to guide the search and uncover effective yet non-obvious combinations of front-end, branch prediction, and memory-system resources.

5 Conclusion

In this paper, we have presented MicroEvo, a knowledge-guided LLM-MCTS framework for microarchitecture DSE. By integrating Pareto-aware tree search, experience accumulation, and state-aware expansion, MicroEvo outperforms prior methods in Pareto quality and sampling efficiency, and remains practical under a complex industrial-grade core. These results suggest that the value of LLMs in microarchitecture multi-objective optimization lies in turning evaluation feedback into more effective exploration decisions.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (Grant No.92464301, U25A6023), the National Key Research and Development Program (Grant No.2024YFB4405600), and the Key Research and Development Program of Jiangsu Province (Grant No.BG2024010).

References

[1] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. 2019. Edge Computing for Autonomous Driving: Opportunities and Challenges. *Proc. IEEE* 107, 8 (2019), 1697–1716.

[2] Matthew Halpern, Yuhao Zhu, and Vijay Janapa Reddi. 2016. Mobile CPU's rise to power: Quantifying the impact of generational mobile CPU design trends on performance, energy, and user satisfaction. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 64–76.

[3] Brian Grayson, Jeff Rupley, Gerald Zuraski Zuraski, Eric Quinnell, Daniel A. Jiménez, Tarun Nakra, Paul Kitchin, Ryan Hensley, Edward Brekelbaum, Vikas Sinha, and Ankit Ghiya. 2020. Evolution of the Samsung Exynos CPU Microarchitecture. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*. 40–51.

[4] Chen Bai, Jiayi Huang, Xuechao Wei, Yuzhe Ma, Sicheng Li, Hongzhong Zheng, Bei Yu, and Yuan Xie. 2023. ArchExplorer: Microarchitecture Exploration Via Bottleneck Analysis. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '23)*. 268–282.

[5] Xuyang Zhao, Tianning Gao, Zheng Wu, Zhaori Bi, Changhao Yan, Fan Yang, Sheng-Guo Wang, Dian Zhou, and Xuan Zeng. 2025. APPLE-DSE: Asynchronous Parallel Pareto Set Learning for Microarchitecture Design Space Exploration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 44, 7 (2025), 2765–2778.

[6] Samuel Naffziger, Noah Beck, Thomas Burd, Kevin Lepak, Gabriel H. Loh, Mahesh Subramony, and Sean White. 2021. Pioneering Chiplet Technology and Design for the AMD EPYC™ and Ryzen™ Processor Families : Industrial Product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*. 57–70.

[7] Tejas S. Karkhanis and James E. Smith. 2007. Automated design of application specific superscalar processors: an analytical approach. In *Proceedings of the 34th Annual International Symposium on Computer Architecture (ISCA '07)*. 402–411.

[8] Guangyu Sun, Christopher Hughes, Changkyu Kim, Jishen Zhao, Cong Xu, Yuan Xie, and Yen-Kuang Chen. 2011. Moguls: A model to explore the memory hierarchy for bandwidth improvements. In *2011 38th Annual International Symposium on Computer Architecture (ISCA)*. 377–388.

[9] Chen Bai, Qi Sun, Jianwang Zhai, Yuzhe Ma, Bei Yu, and Martin D.F. Wong. 2021. BOOM-Explorer: RISC-V BOOM Microarchitecture Design Space Exploration Framework. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–9.

[10] Duo Wang, Mingyu Yan, Xin Liu, Mo Zou, Tianyu Liu, Wenming Li, Xiaochun Ye, and Dongrui Fan. 2023. A High-accurate Multi-objective Exploration Framework for Design Space of CPU. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. 1–6.

[11] Runzhen Xue, Hao Wu, Mingyu Yan, Ziheng Xiao, Xiaochun Ye, and Dongrui Fan. 2025. MetaDSE: A Few-Shot Meta-Learning Framework for Cross-Workload CPU Design Space Exploration. In *Proceedings of the 62nd Annual ACM/IEEE Design Automation Conference (DAC '25)*. 7 pages.

[12] Ziyang Yu, Chen Bai, Shoubo Hu, Ran Chen, Taohai He, Mingxuan Yuan, Bei Yu, and Martin Wong. 2023. IT-DSE: Invariance Risk Minimized Transfer Microarchitecture Design Space Exploration. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9.

[13] Duo Wang, Mingyu Yan, Yihan Teng, Dengke Han, Haoran Dang, Xiaochun Ye, and Dongrui Fan. 2023. A Transfer Learning Framework for High-Accurate Cross-Workload Design Space Exploration of CPU. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. 1–9.

[14] Jianwang Zhai and Yici Cai. 2023. Microarchitecture Design Space Exploration via Pareto-Driven Active Learning. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 31, 11 (2023), 1727–1739.

[15] Xingyu Wu, Sheng-Hao Wu, Jibin Wu, Liang Feng, and Kay Chen Tan. 2025. Evolutionary Computation in the Era of Large Language Model: Survey and Roadmap. *IEEE Transactions on Evolutionary Computation* 29, 2 (2025), 534–554.

[16] Jingyuan Li, Jianrong Zhang, Ye Li, Wenbo Yin, and Lingli Wang. 2025. LEMOE: LLM-Enhanced Multi-Objective Bayesian Optimization for Microarchitecture Exploration. In *2025 62nd ACM/IEEE Design Automation Conference (DAC)*. 1–7.

[17] Zhantong Zhu, Kangbo Bai, and Tianyu Jia. 2026. Chat-A2: An LLM-aided Design Space Exploration Framework for High-Performance CPU Design. In *2026 31st Asia and South Pacific Design Automation Conference (ASP-DAC)*. 540–546.

[18] Zheng Wu, Zhuochu Yang, Zhuoyuan Yang, Zihao Chen, Li Shang, and Fan Yang. 2025. ChatArch: A Knowledge-driven Graph-of-thought LLM Framework for Processor Architecture Optimization. *ACM Trans. Des. Autom. Electron. Syst.* 31, 2 (2025), 26 pages.

[19] Mingxin Tang, Wei Chen, Lizhou Wu, Libo Huang, and Kun Zeng. 2025. ChatDSE: A Zero-Shot Microarchitecture Design Space Explorer Powered by GPT4.0. *ACM Trans. Des. Autom. Electron. Syst.* 30, 4 (2025), 24 pages.

[20] Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. 2024. Evolution of heuristics: towards efficient automatic algorithm design using large language model. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*. 23 pages.

[21] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. 2024. Mathematical discoveries from program search with large language models. *Nature* 625, 7995 (2024), 468–475.

[22] Zhi Zheng, Zhuoliang Xie, Zhenkun Wang, and Bryan Hooi. 2025. Monte Carlo Tree Search for Comprehensive Exploration in LLM-Based Automatic Heuristic Design. In *International Conference on Machine Learning*. 78338–78373.

[23] Haoran Ye, Jiarui Wang, Zhiguang Cao, Federico Berto, Chuanbo Hua, Haeyeon Kim, Jinkyoo Park, and Guojie Song. 2024. ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution. In *Advances in Neural Information Processing Systems*, Vol. 37. 43571–43608.

[24] Johnson Umeike, Neel Patel, Alex Manley, Amin Mamandipoor, Heechul Yun, and Mohammad Alian. 2023. Profiling gem5 Simulator. In *2023 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 103–113.

[25] Andreia P. Guerreiro, Carlos M. Fonseca, and Luís Paquete. 2021. The Hypervolume Indicator: Computational Problems and Algorithms. *ACM Comput. Surv.* 54, 6 (2021), 42 pages.

[26] Chen Bai, Jianwang Zhai, Yuzhe Ma, Bei Yu, and Martin D. F. Wong. 2024. Towards Automated RISC-V Microarchitecture Design with Reinforcement Learning. *Proceedings of the AAAI Conference on Artificial Intelligence* 38, 1 (2024), 12–20.

[27] Changwen Xing, SamZaak Wong, Xinlai Wan, Yanfeng Lu, Mengli Zhang, Zebin Ma, Lei Qi, Zhengxiong Li, Nan Guan, Zhe Jiang, Xi Wang, and Jun Yang. 2026. ChipMind: Retrieval-Augmented Reasoning for Long-Context Circuit Design Specifications. *Proceedings of the AAAI Conference on Artificial Intelligence* 40, 2 (2026), 1337–1345.

[28] Xi Wang, Gwok-Waa Wan, Sam-Zaak Wong, Layton Zhang, Tianyang Liu, Qi Tian, and Jianmin Ye. 2024. ChatCPU: An Agile CPU Design and Verification Platform with LLM. In *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*. 6 pages.

[29] Lik Tung Fu, Jie Zhou, Shaokai Ren, Mengli Zhang, Jia Xiong, Hugo Jiang, Nan Guan, Xi Wang, and Jun Yang. 2026. ChatSVA: Bridging SVA Generation for Hardware Verification via Task-Specific LLMs. *arXiv preprint arXiv:2604.02811* (2026).

[30] Runkai Li, Jia Xiong, Xiuyuan He, Jieru Zhao, Jiaqi Lv, Haowen Fang, Lei Qi, and Xi Wang. 2026. ChatHLS: Towards Systematic Design Automation and Optimization for High-Level Synthesis. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 20996–21015.

[31] Gwok-Waa Wan, SamZaak Wong, Shengchu Su, Chenxu Niu, Ning Wang, Xinlai Wan, Qixiang Chen, Mengnv Xing, Jingyi Zhang, Jianmin Ye, et al. 2026. FIXME: Towards End-to-End Benchmarking of LLM-Aided Design Verification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 40. 1087–1095.

[32] Runkai Li, Jia Xiong, and Xi Wang. 2025. iDSE: Navigating Design Space Exploration in High-Level Synthesis Using LLMs. *arXiv preprint arXiv:2505.22086* (2025).

[33] Jianmin Ye, Tianyang Liu, Qi Tian, Shengchu Su, Zhe Jiang, and Xi Wang. 2025. ChatModel: Automating Reference Model Design and Verification with LLMs. *arXiv preprint arXiv:2506.15066* (2025).

[34] Maciej Świechowski, Konrad Godlewski, Bartosz Sawicki, and Jacek Mańdziuk. 2023. Monte Carlo tree search: A review of recent modifications and applications. *Artificial Intelligence Review* 56, 3 (2023), 2497–2562.

[35] Yong Hu, Daniel Mueller-Gritschneder, and Ulf Schlichtmann. 2018. Wavefront-MCTS: Multi-objective Design Space Exploration of NoC Architectures based on Monte Carlo Tree Search. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8.

[36] Hui Wang, Yang Liu, Xiaoyu Zhang, and Chaoxu Mu. 2025. CogMCTS: A Novel Cognitive-Guided Monte Carlo Tree Search Framework for Iterative Heuristic Evolution with Large Language Models. *arXiv preprint arXiv:2512.08609* (2025).

[37] Chentong Chen, Mengyuan Zhong, Ye Fan, Jialong Shi, and Jianyong Sun. 2025. HiFo-Prompt: Prompting with Hindsight and Foresight for LLM-based Automatic Heuristic Design. *arXiv preprint arXiv:2508.13333* (2025).

[38] Yoshihiko Ozaki, Yuki Tanigaki, Shuhei Watanabe, and Masaki Onishi. 2020. Multiobjective tree-structured parzen estimator for computationally expensive optimization problems. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference (GECCO '20)*. 533–541.

[39] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The gem5 simulator. *SIGARCH Comput. Archit. News* 39, 2 (2011), 1–7.

[40] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. 2009. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 42)*. 469–480.

[41] 2026. Official RISC-V Benchmark Suites. https://github.com/riscv-software-src/riscv-tests

[42] Kaifan Wang, Jian Chen, Yinan Xu, Zihao Yu, Wei He, Dan Tang, Ninghui Sun, and Yungang Bao. 2025. XiangShan: An Open Source Project for High-Performance RISC-V Processors Meeting Industrial-Grade Standards. *IEEE Micro* 45, 3 (2025), 49–57.

[43] 2026. XS-GEM5. https://github.com/OpenXiangShan/GEM5