ORDINALFORGE: A SCORE-INDUCED RANK REPRESENTATION FOR SCALABLE TRAINING-FREE DNN–HARDWARE CO-OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

As modern edge AI applications continue to grow in computational complexity, co-optimizing Deep Neural Networks (DNNs) and their underlying hardware becomes increasingly critical. Existing Neural Architecture Search (NAS) approaches face two major challenges: the high dimensionality of the joint DNN–hardware representation and the substantial costs of evaluating DNNs through training. Particularly, while training-free NAS can reduce evaluation costs, using the proxy score as the DNN objective alone does not reduce the dimensionality of the surrogate input. To address this limitation, we introduce OrdinalForge, a training-free DNN–hardware co-optimization framework using a score-induced rank formulation for surrogate-based optimization. By sorting the proxy scores and assigning each DNN candidate a scalar rank, OrdinalForge replaces the original multidimensional DNN candidates supplied to the surrogate with a one-dimensional ordered input, allowing the surrogate to model the proxy score as a monotonic function of rank. Experiments show that OrdinalForge improves hypervolume by up to $13.7\times$ and reduces search time by up to $57.8\times$ compared to training-based baselines in image classification and semantic segmentation workloads. It also achieves the highest mean final cumulative hypervolume and better Pareto-optimal solutions among the evaluated methods in image classification, semantic segmentation, and RNN-based language modeling tasks. In the GPT-2-based language processing workload, the score-induced rank formulation improves the mean final cumulative hypervolume compared to its corresponding original-encoding baselines, validating that the proposed approach is effective, generalizable, and scalable.

1 INTRODUCTION

Deep Neural Networks (DNNs) have been deployed across a wide range of applications, including semantic segmentation (Yan et al., 2023), natural language processing (NLP) (Liu et al., 2024), and recommendation systems (Covington et al., 2016; He et al., 2017). However, as the number of parameters in modern DNNs continues to surge, the associated computational cost grows substantially (Desislavov et al., 2023). To improve the efficiency of AI execution, it is crucial to balance model performance with hardware resources, particularly for edge AI accelerators where the power and area budgets are stringent (Lu et al., 2024; Li et al., 2024; 2023a; Zhang et al., 2023; Wang et al., 2019).

To address this challenge, methods for co-optimizing DNN performance and the underlying hardware metrics have been proposed. These co-optimization strategies are categorized into single-objective and multi-objective approaches (Krestinskaya et al., 2024). Single-objective approaches can be further divided into two-stage methods and constrained methods. Two-stage methods implement two optimization algorithms where the first searches for optimal DNN architectures to maximize performance, while the second addresses hardware constraints to ensure the efficiency (Cai et al., 2020). However, the identified DNNs do not always satisfy the hardware constraints, as the two optimization stages are separate. Constrained methods solve this issue by jointly considering hardware constraints and DNN performance when updating DNN architectures (Tan et al., 2019).

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

However, the optimization of hardware architectures is not directly integrated into this process, as these algorithms merely update network architectures.

In contrast, multi-objective approaches formulate the co-optimization problem by considering multiple objectives related to DNN performance and hardware efficiency. The scalarization method is a prominent technique in multi-objective optimization that integrates various objectives associated with DNNs and hardware into a single objective through a weighted sum or product (Hsu et al., 2018). Another widely adopted approach is Pareto optimization, which seeks to obtain a set of solutions that are non-dominated with respect to multiple objectives. Popular algorithms for Pareto optimization include the Non-dominated Sorting Genetic Algorithm II (NSGA-II) (Deb et al., 2002) and multi-objective Bayesian optimization (MOBO) (Krestinskaya et al., 2024; Parsa et al., 2019; Sekanina, 2021).

Particularly, the multi-objective approaches enable the discovery of non-dominated design points that more effectively capture trade-offs across conflicting objectives compared to single-objective methods (Das et al., 2024). Jiang et al. (2020) highlight the increasing prominence of multi-objective optimization, demonstrating that it can achieve significantly higher energy efficiency without compromising DNN performance. However, these approaches are inherently more complex and computationally intensive than single-objective methods. The joint DNN–hardware design space in multi-objective formulations is typically high-dimensional, as it spans a vast array of parameters across both the neural network architecture and the hardware configurations. The high dimensionality poses a significant challenge (Binois & Wycoff, 2022; Tian et al., 2021), particularly in the DNN design space, where architectural complexity has grown exponentially in recent years (Cheng et al., 2024). Furthermore, the evaluation cost for candidate DNN architectures (i.e., training time) dominates the total search overhead in multi-objective DNN–hardware co-optimization (Benmeziane et al., 2023).

Training-free Neural Architecture Search (NAS) is a promising technique for reducing the cost of evaluating candidate DNNs by estimating their relative performance using information obtained from untrained networks, avoiding the costly training and validation of each candidate network (Yamasaki et al., 2025; Mellor et al., 2021; Chen et al., 2021; Lee et al., 2019; Tanaka et al., 2020; Li et al., 2023b; Abdelfattah et al., 2021). However, in the context of DNN–hardware co-optimization, direct-score MOBO using the RBFleX-NAS (Yamasaki et al., 2025) proxy with the original multidimensional DNN encoding achieves a lower mean final cumulative hypervolume than accuracy-based MOBO, as shown in Table 1.

Table 1: Final cumulative hypervolume of the evaluated baseline methods in the image classification setting over three independent runs using seeds 42, 1234, and 2026. Results are reported as mean $\pm$ sample standard deviation and CV, defined as the sample standard deviation divided by the mean. Initial designs are sampled independently for each method and seed.

Method	**Runs**	**Final cumulative hypervolume Mean $\pm$ sample standard deviation**	**CV (%)**
MOBO (RBFleX-NAS score, original DNN encoding)	3	$1.37 \times 10^7 \pm 3.34 \times 10^5$	2.44
NSGA-II (Accuracy metric)	3	$1.31 \times 10^7 \pm 6.39 \times 10^5$	4.86
Random Search (Accuracy metric)	3	$1.30 \times 10^7 \pm 2.08 \times 10^5$	1.60
MOBO (Accuracy metric)	3	$1.49 \times 10^7 \pm 1.91 \times 10^5$	1.29

To address this degradation in co-optimization performance, we introduce OrdinalForge, a training-free framework for jointly optimizing DNN architectures and hardware accelerator design. OrdinalForge employs a score-induced rank formulation that replaces the multidimensional software-side component of the surrogate input with a scalar rank. The framework adopts a divide-and-conquer strategy that decouples the evaluation of DNN and hardware configuration candidates. Specifically, it precomputes DNN performance scores using a training-free NAS proxy and stores each candidate DNN together with its proxy score in a ranking table. OrdinalForge addresses these two challenges as follows. First, by precomputing training-free proxy scores, OrdinalForge replaces repeated training-based DNN evaluation during search with proxy score retrieval. Second, this decomposition significantly reduces the dimensionality of the joint input representation supplied to the optimizer.

Moreover, OrdinalForge systematically incorporates the ranking table into the optimization process by conditioning hardware candidate updates on pre-ranked DNN architectures. This modular design supports efficient DNN–hardware co-optimization over the evaluated finite candidate spaces. OrdinalForge achieves up to $13.7\times$ higher hypervolume and up to a $57.8\times$ speedup over the evaluated training-based baselines in image classification and semantic segmentation tasks. Across the multi-seed evaluations, it achieves the highest mean final cumulative hypervolume among the evaluated approaches in image classification (1.52×10^7), semantic segmentation (6.21×10^5), and RNN-based language modeling workloads (1.120). In GPT2-based causal language processing tasks, the score-induced rank formulations leveraging the state-of-the-art training-free NAS (RBFleX-NAS (Yamasaki et al., 2025) and NASWOT (Mellor et al., 2021)) improve the mean final cumulative hypervolume over their corresponding original-encoding baselines by 2.10% and 3.22%, respectively. These results show that the benefit of the score-induced rank is effective, scalable, and generalizable. The OrdinalForge framework is publicly available at `https://anonymous.4open.science/r/H-RBFleX-FCD4/`

2 METHOD

2.1 PREPARATION

Compared with single-objective DNN optimization, a method that jointly optimizes DNNs and hardware accelerators operates over a higher-dimensional input space because it additionally includes hardware design parameters. Such a method needs to generate candidate solutions including, for example, the number of layers, kernel sizes, and activation functions in the DNN design space, as well as the number of PEs, memory size, and dataflow configurations in the hardware design space. If the DNN design space has n dimensions and the hardware design space has m dimensions, the joint design space $\mathcal{D}$ can be expressed as follows:

$$\mathcal{D} = \mathcal{D}_s \times \mathcal{D}_h \subseteq \mathbb{R}^n \times \mathbb{R}^m = \mathbb{R}^{n+m}. \tag{1}$$

where $\mathcal{D}_s \subseteq \mathbb{R}^n$ denotes the n-dimensional DNN design space, and $\mathcal{D}_h \subseteq \mathbb{R}^m$ denotes the m-dimensional hardware design space.

To construct the score-induced rank representation, OrdinalForge assigns each DNN candidate in $\mathcal{D}_s$ a score using a training-free NAS proxy and sorts the candidates in descending order of their scores. As illustrated in Fig. 1(a), the resulting ranking table maps each n-dimensional DNN configuration $\mathbf{x}_s \in \mathcal{D}_s \subseteq \mathbb{R}^n$ to a scalar rank $r \in \mathcal{R}_s$, reducing the software-side representation supplied to the optimizer from n dimensions to one. After all candidates in $\mathcal{D}_s$ are scored and ranked, the reduced joint input domain is defined as:

$$\mathcal{D}' = \mathcal{R}_s \times \mathcal{D}_h \subseteq \mathbb{R} \times \mathbb{R}^m = \mathbb{R}^{1+m}. \tag{2}$$

Here, $\mathcal{R}_s = \{1, \ldots, |\mathcal{D}_s|\}$ denotes the set of valid DNN ranks, and $\mathcal{D}_h \subseteq \mathbb{R}^m$ denotes the hardware design space. The original and reduced joint input dimensions are $d_{\text{in}} = n + m$ and $d'_{\text{in}} = 1 + m$, respectively. Their ratio therefore becomes

$$\frac{d'_{\text{in}}}{d_{\text{in}}} = \frac{1+m}{n+m}. \tag{3}$$

The ranking table maintains a one-to-one mapping between each rank and a DNN candidate, allowing the corresponding DNN configuration to be retrieved for hardware evaluation. The surrogate operates on the lower-dimensional pair $(r, \mathbf{x}_h)$.

2.2 CO-OPTIMIZATION

OrdinalForge subsequently performs co-optimization to identify favorable combinations of DNN architectures and hardware configurations. Fig. 1 (b) illustrates the co-optimization workflow. During this phase, OrdinalForge generates candidate DNNs characterized by a single parameter, and

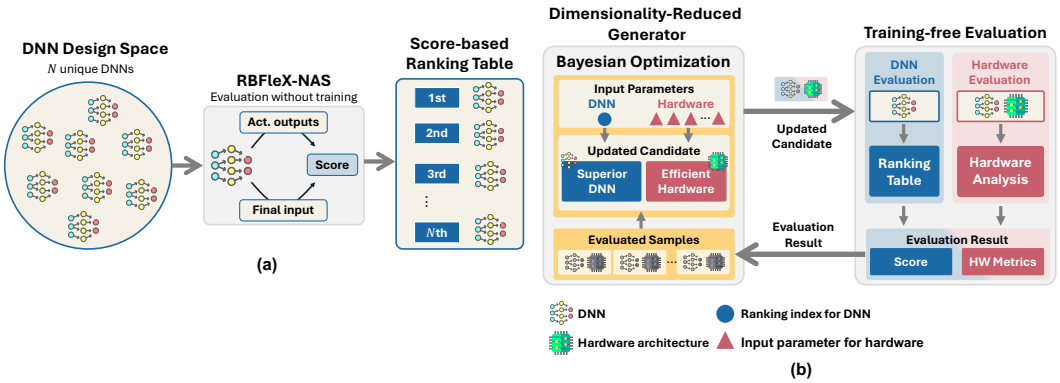


Figure 1: Overview of OrdinalForge: (a) OrdinalForge constructs a score-induced ranking table from training-free proxy scores. (b) MOBO performs DNN–hardware co-optimization in the reduced joint space using the DNN rank and hardware parameters.

hardware architectures defined by m hardware parameters. The precomputed ranking table provides the proxy score for each candidate DNN, while the hardware analysis tool evaluates the corresponding DNN–hardware pair to obtain hardware metrics. For comparison, a conventional Pareto multi-objective co-optimization method optimizes the vector-valued objective $\boldsymbol{f}(\boldsymbol{X}) \in \mathbb{R}^{1+M}$, which consists of a DNN performance objective $f_s(\boldsymbol{x}_s)$ and M hardware metrics $f_h(\boldsymbol{X})$, where $\boldsymbol{X} = [\boldsymbol{x}_s, \boldsymbol{x}_h] \in \mathbb{R}^{n+m}$ is the joint input vector, as shown by

$$\boldsymbol{f}(\boldsymbol{X}) = [f_s(\boldsymbol{x}_s), f_h^1(\boldsymbol{X}), \ldots, f_h^M(\boldsymbol{X})] \tag{4}$$

where $\boldsymbol{x}_s$ and $\boldsymbol{x}_h$ indicate configurations of a DNN and a hardware architecture, respectively. In contrast, OrdinalForge uses a multi-objective optimization algorithm to optimize the objective vector defined in Equation (5)

$$\boldsymbol{f}(\boldsymbol{X'}) = [\rho(r), f_h^1(\boldsymbol{X'}), \ldots, f_h^M(\boldsymbol{X'})] \tag{5}$$

where $r \in \mathcal{R}_s$ is the rank index assigned to a candidate DNN, $\rho(r)$ is its proxy score retrieved from the ranking table, and the input vector of OrdinalForge is $\boldsymbol{X'} = [r, \boldsymbol{x}_h] \in \mathbb{R}^{1+m}$. Let $\rho(\boldsymbol{x}_s)$ denote the training-free proxy score of a DNN architecture $\boldsymbol{x}_s \in \mathcal{D}_s$. We sort all candidate architectures in descending order of ρ and assign each architecture a unique rank $r(\boldsymbol{x}_s) \in \{1, \ldots, |\mathcal{D}_s|\}$. Specifically, the candidate with the highest proxy score is assigned a rank of 1. Candidates with equal proxy scores are ordered deterministically using their original architecture indices.

Moreover, Yamasaki et al. (2025) report positive ranking correlations between the proxy scores produced by its training-free NAS, RBFlex-NAS and final trained performance across multiple NAS benchmarks. Specifically, on NATS-Bench-SSS (Dong et al., 2021), a Kendall correlation of 0.639 on CIFAR-100 and 0.649 on ImageNet-16-120 is reported. On TransNAS-Bench-101 for semantic segmentation, a Kendall correlation of 0.668 is achieved. These correlations quantify the agreement between the score-induced ordering and post-training performance, which motivates us to use score-based ranking for OrdinalForge.

2.3 MOBO Procedure in the Rank-Based Design Space

OrdinalForge adopts a MOBO strategy, which uses surrogate models and an Acquisition Function (AF) to guide candidate selection from the accumulated observations. After constructing the rank-based joint input $\boldsymbol{X'} = [r, \boldsymbol{x}_h]$, OrdinalForge applies MOBO to search the joint design space (Fig. 1 (b)). At iteration t, let $\boldsymbol{X}'_{t,j}$ denote the candidate selected for the jth position in the next batch, where $j \in \{1, \ldots, n_b\}$ and n_b is the batch size. Each candidate is selected according to

$$\boldsymbol{X}'_{t,j} \in \arg\max_{\boldsymbol{X'} \in \mathcal{D'}} \alpha_{t,j}\left(\boldsymbol{X'}; \mathcal{H}_{t-1}, \boldsymbol{X}'_{t,<j}\right), \tag{6}$$

where $\mathcal{D}' \subset \mathbb{R}^{1+m}$ denotes the feasible joint design space of $\boldsymbol{X}' = [r, \boldsymbol{x}_h]$. The set $\boldsymbol{X}'_{t,<j} = \{\boldsymbol{X}'_{t,1}, \ldots, \boldsymbol{X}'_{t,j-1}\}$ contains the candidates already selected for the current batch, with $\boldsymbol{X}'_{t,<1} = \emptyset$. The resulting candidate batch is $\boldsymbol{X}'_t = \{\boldsymbol{X}'_{t,1}, \ldots, \boldsymbol{X}'_{t,n_b}\}$. For each batch position j, the AF $\alpha_{t,j}$ is computed from the Gaussian Process (GP) posterior conditioned on the evaluation history $\mathcal{H}_{t-1}$, while the previously selected candidates $\boldsymbol{X}'_{t,<j}$ are treated as pending. The mean and uncertainty of the GP posterior allow the AF to balance exploitation and exploration over the joint design space $\mathcal{D}'$. In our implementation, qNParEGO (Daulton et al., 2020) serves as the acquisition strategy for constructing these position-specific AFs. It combines randomized augmented Chebyshev scalarizations with noisy expected improvement, which estimates the candidate's expected improvement by integrating over posterior uncertainty in the objective values of both the candidate and the previously evaluated points.

At each batch position, qNParEGO samples a new weight vector for the augmented Chebyshev scalarization to explore a different trade-off among the objectives. Precomputing the proxy scores removes repeated DNN evaluation during optimization. For candidate selection, qNParEGO uses GP models fitted to the accumulated evaluation history $\mathcal{H}_{t-1}$. Their posterior predictions over the joint input $\boldsymbol{X}' = [r, \boldsymbol{x}_h]$ are used to select the next candidate batch $\boldsymbol{X}'_t$. After selection, OrdinalForge retrieves the exact precomputed proxy score $\rho(r)$ from the ranking table and obtains the hardware objective values $\boldsymbol{f}_h(\boldsymbol{X}')$ from the hardware analysis tool. Therefore, the GP models guide acquisition while the ranking table and hardware evaluator supply the objective values recorded for the selected candidates. These observations are incorporated into the evaluation history for $t \in \{1, 2, \ldots, T\}$, where T is the total number of iterations.

3 Experimental Results

In this section, we evaluate OrdinalForge through four experiments: image classification on NATS-Bench-SSS (Dong et al., 2021), semantic segmentation on TransNAS-Bench-101-Micro (Duan et al., 2021), recurrent language modeling on NAS-Bench-NLP (Klyuchnikov et al., 2022), and causal language modeling with GPT-S on HW-GPT-Bench (Sukthanker et al., 2024). For the first three workloads, each optimization method begins with 100 initial designs generated using Sobol sampling and then runs 30 iterations with 10 newly generated candidates per iteration. The causal language modeling setting uses the same total evaluation budget, but all methods and search seeds share the same 100 canonical initial design pairs.

3.1 Analysis and Comparison of Multi-objective Optimizations for Image Classification

In this experiment, we evaluate the efficiency of OrdinalForge for image classification by comparing it with five multi-objective optimization baselines. To benchmark with accuracy-based approaches, where neural network performance is assessed through training, we compare OrdinalForge against the implementaions by using (1) MOBO, (2) Random Search, and (3) NSGA-II. To compare with NAS-score-based methods, which do not require network training, we compare it agains the implementations by using (4) MOBO with RBFleX-NAS and (5) MOBO with a pre-computed ranking table using NASWOT (Mellor et al., 2021). We adopt NATS-Bench-SSS (Dong et al., 2021) as the DNN design space. For hardware evaluation, we integrate SCALE-Sim (Samajdar et al., 2020) to estimate the cycle count for model execution.

Fig. 2 presents the optimization results in terms of hypervolume and search cost, using a reference point of a classification error of 70% and a cycle count of 1.0×10^6. In the run shown, OrdinalForge achieves the highest hypervolume of 1.51×10^7 with the least total search time of 32.9 hours, including 0.98 hours for preparation and 31.9 hours for co-optimization. Using the settings in Table 1, we compare OrdinalForge against the direct-score MOBO while keeping the same scoring method. OrdinalForge attains a final cumulative hypervolume of $1.52 \times 10^7 \pm 3.47 \times 10^3$, while the direct-score MOBO obtains a final cumulative hypervolume of $1.37 \times 10^7 \pm 3.34 \times 10^5$. OrdinalForge achieves a higher value in all three runs and improves the mean value by 10.71%. To compare the optimization behavior of these two formulations, Fig. 3 shows that the direct-score MOBO provides sub-optimal solutions for DNN–hardware co-design compared to OrdinalForge. We also evaluate accuracy-based MOBO and NSGA-II under the same three seeds and final hypervolume protocol.

OrdinalForge improves the mean final cumulative hypervolumes by 2.07% and 15.31%, respectively, and achieves the highest mean final cumulative hypervolume among all evaluated methods.

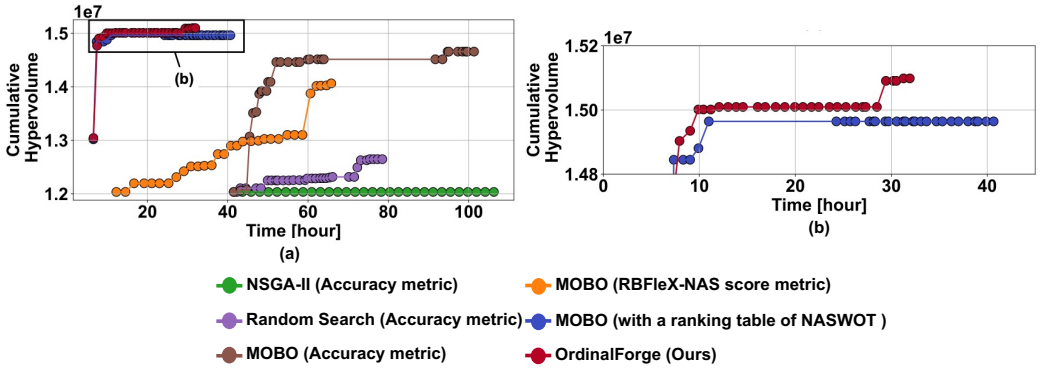


Figure 2: Hypervolume progression in the image classification setting with a reference point of 70% classification error and 1.0×10^6 cycles. Accuracy-based baselines use DNN accuracy measured after 90 training epochs. (a) Comparison with the evaluated optimization baselines. (b) Comparison with MOBO using a NASWOT ranking table.

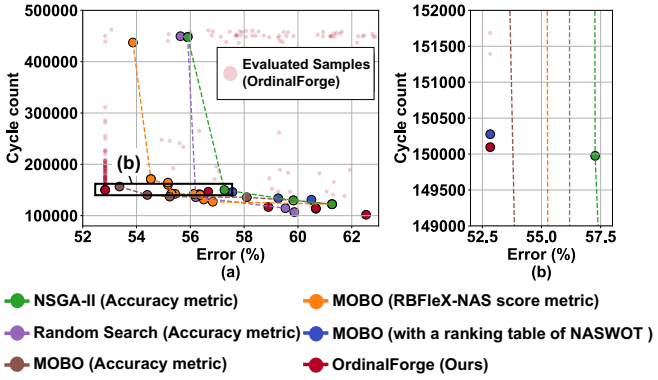


Figure 3: (a) Optimization results for cycle count versus ImageNet-16-120 error (%) on NATS-Bench-SSS. For NSGA-II, random search, and MOBO, candidate updates use DNN accuracy measured after 90 training epochs. (b) Enlarged view of the region enclosed by the black box in (a).

3.2 Comparative Analysis of Multi-objective Optimizations for Semantic Segmentation

In this experiment, we evaluate OrdinalForge and the five co-optimization baselines in the semantic segmentation setting. We use TransNAS-Bench-101-Micro, which contains 4,096 DNN architectures evaluated on the Taskonomy dataset (Zamir et al., 2018). The semantic segmentation error is defined as $100\% - \mathrm{mIoU}$. The remaining settings are the same as those used for image classification. As shown in Fig. 4, OrdinalForge achieves a final hypervolume of 6.83×10^5, corresponding to improvements of $3.7\times$, $13.7\times$, and $7.3\times$ over mIoU-based MOBO, NSGA-II, and random search, respectively.

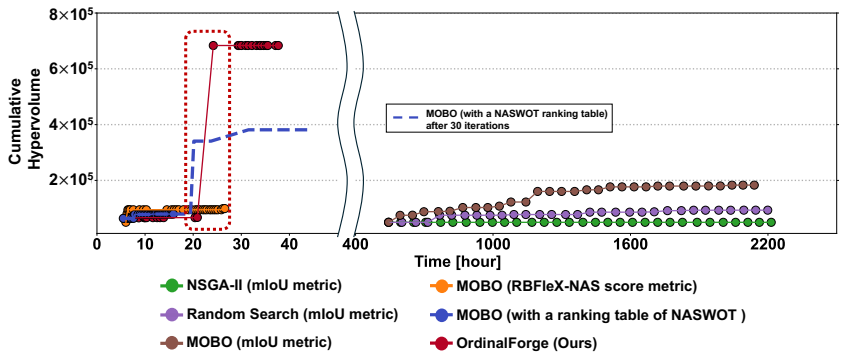


Figure 4: Hypervolume progression in the semantic segmentation setting. The reference point is set to a semantic segmentation error of 85% and a cycle count of 1.0×10^5. The mIoU-based NSGA-II, random search, and MOBO baselines update candidates using DNN performance measured after 30 training epochs. These mIoU-based methods require more than 400 hours to evaluate the 100 Sobol-sampled initial designs. The dashed black segment shows the continuation of MOBO with a NASWOT ranking table beyond its 30th iteration. Each curve reports one run of the corresponding method.

Moreover, these baselines require training every candidate DNN, resulting in search time that is approximately $56.3\times$ to $57.8\times$ longer than that of OrdinalForge. Particularly, they require more than 400 hours to evaluate 100 Sobol-sampled initial designs. The complete Pareto fronts are shown in Fig. 5.

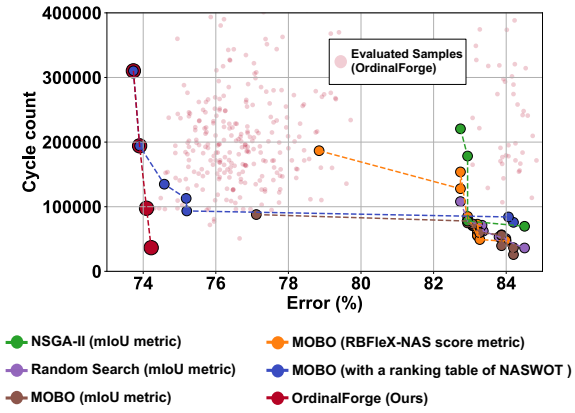


Figure 5: Optimization results for cycle count versus semantic segmentation error (%) on TransNAS-Bench-101-Micro. For NSGA-II, random search, and MOBO, candidates are updated using mIoU measured after 30 training epochs.

3.3 Comparative Analysis of Multi-Objective Optimization Methods for RNN-based Language Modeling

We apply OrdinalForge to the NAS-Bench-NLP design space and use perplexity as the network model metric and Energy-Delay Product (EDP) as the hardware metric. This recurrent neural network (RNN) design space incorporates 14,322 unique network architectures trained on the Penn TreeBank (PTB) dataset (Mikolov et al., 2010) as ground truth. In this workload, we use an analytical EDP inspired by accelerator cost-modeling concepts in MAESTRO (Kwon et al., 2020) and Eyeriss (Chen et al., 2016).

As Fig. 6 shows, OrdinalForge completes the search in 0.4 hours, whereas the training-based MOBO baseline requires about 1454 hours, resulting in a $3635\times$ search time reduction. The runs yield a hypervolume of 0.45 in 0.26 hours for the MOBO using RBFleX-NAS scores, 1.11 in 0.42 hours for

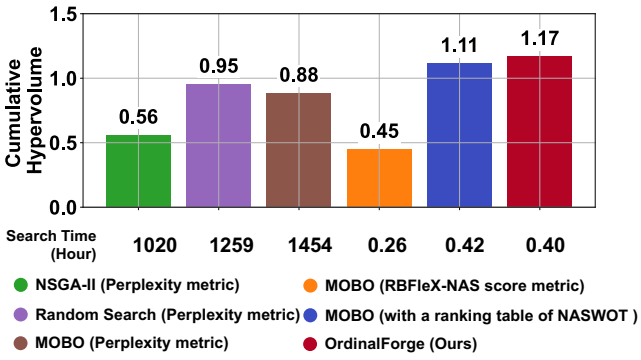


Figure 6: Hypervolume after 30 iterations in the RNN-based language modeling setting under the shared analytical EDP proxy. The reference point is 1.8 for normalized perplexity and 2.5 for the normalized analytical EDP proxy. Both objectives are normalized using fixed factors shared by all methods and seeds.

the MOBO with a NASWOT ranking table, and 1.17 in 0.4 hours for OrdinalForge. OrdinalForge achieves the highest hypervolume among these three training-free methods. Across five independent runs, our approach achieves a final cumulative hypervolume of 1.120 ± 0.106, which is 34.37% higher on average than the perplexity-based MOBO. Fig. 7 shows the achieved Pareto-optimal solutions that trade off between normalized perplexity and the normalized analytical EDP proxy.

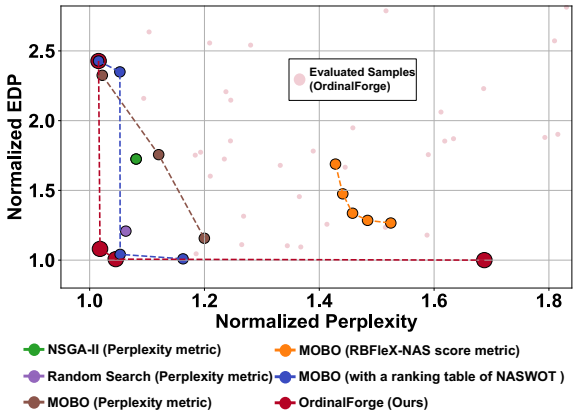


Figure 7: Optimization results for the normalized analytical EDP proxy versus normalized perplexity in the recurrent language modeling setting. For NSGA-II, random search, and perplexity-based MOBO, candidates are updated using DNN perplexity measured after 50 training epochs. Both objectives are normalized using fixed factors shared by all methods and seeds.

3.4 Comparative Analysis of Multi-Objective Optimization Methods for Causal Language Modeling

We further evaluate the ranking-based formulation on the GPT-S architecture space in HW-GPT-Bench, using perplexity and inference latency as metrics. The latency is estimated by H^2-LLM (Li et al., 2025). We conduct two controlled comparisons. The first compares OrdinalForge with the MOBO using the RBFleX-NAS score and the original GPT-S encoding. The second compares the MOBO using a NASWOT ranking table against the MOBO using the NASWOT score and the original GPT-S encoding. Within each comparison, the proxy score, candidate domain, hardware objective, optimizer configuration, evaluation budget, and hypervolume evaluation protocol are fixed. As shown in Fig. 8, across five optimization runs, OrdinalForge achieves a comparable mean final

cumulative hypervolume as the MOBO with a NASWOT ranking table, higher than other baselines.

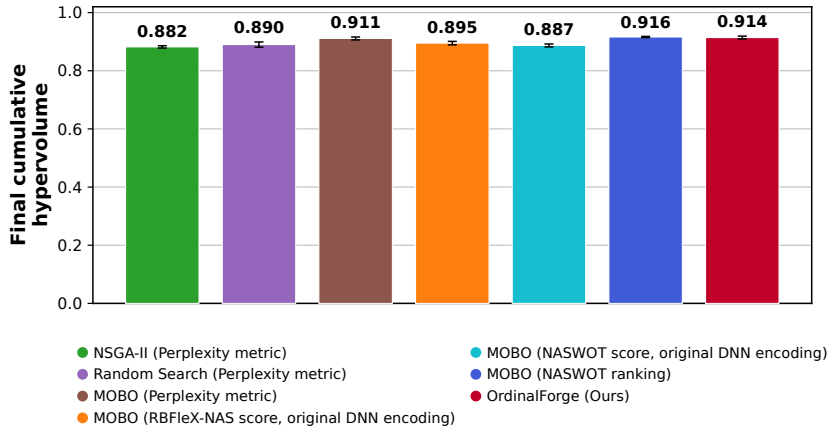


Figure 8: Final cumulative hypervolume in the causal language modeling setting over five independent runs. Bars show the mean and error bars indicate the sample standard deviation.

OrdinalForge improves the mean final cumulative hypervolume over its original-encoding baseline using the RBFleX-NAS score by 2.10%, while the NASWOT ranking variant improves over its original-encoding baseline using the NASWOT score by 3.22%. Each ranking-based formulation achieves a higher final cumulative hypervolume than its corresponding original-encoding baseline for all five search runs. These results show that the score-induced rank is effective and generalizable to large language model processing.

4 Conclusion

In this work, we introduce OrdinalForge, a training-free framework for joint optimization of DNN architectures and hardware accelerators. OrdinalForge uses a score-induced rank formulation for surrogate-based optimization. It decomposes the Pareto-based optimization into a preparation phase and a co-optimization phase, enabling training-free DNN evaluation through precomputed proxy scores and reducing the dimensionality of the joint input representation. On image classification and semantic segmentation, OrdinalForge achieves higher hypervolume than the evaluated baselines and identifies favorable trade-offs between DNN performance and hardware efficiency. The language modeling evaluations further show that OrdinalForge is effective and generalizable to RNN and GPT-style architectures, and the results for the causal language processing demonstrate gains over the original-encoding baselines with scores based on state-of-the-art training-free frameworks.

AI use statement

Generative AI tools were used for simulated peer review and language correction. They also provided feedback on experimental protocols, reproducibility, and the interpretation and presentation of author-generated results. The authors executed all experiment code and generated the reported numerical results from those runs. All AI-assisted suggestions were reviewed and verified by the authors. The authors made all final research decisions and take responsibility for the content of this work.

Reproducibility statement

The experimental settings, design spaces, optimization budgets, implementation details, software environment, and random seeds are documented in Section 3 and Appendix A. The appendix also reports multi-seed statistics for the evaluated settings. The anonymized implementation is available through the link provided in Section 1.

REFERENCES

Mohamed S. Abdelfattah, Abhinav Mehrotra, Łukasz Dudziak, and Nicholas D. Lane. Zero-Cost Proxies for Lightweight NAS. In *Proceedings of the 9th International Conference on Learning Representations*, 2021.

Maximilian Balandat, Brian Karrer, Daniel Jiang, Samuel Daulton, Ben Letham, Andrew G Wilson, and Eytan Bakshy. BoTorch: A framework for efficient Monte-Carlo Bayesian optimization. *Advances in Neural Information Processing Systems*, 33:21524–21538, 2020.

Hadjer Benmeziane, Hamza Ouarnoughi, Kaoutar El Maghraoui, and Smail Niar. Multi-objective hardware-aware neural architecture search with Pareto rank-preserving surrogate models. *ACM Trans. Archit. Code Optim.*, 20(2), April 2023. ISSN 1544-3566. doi: 10.1145/3579853. URL https://doi.org/10.1145/3579853.

Mickael Binois and Nathan Wycoff. A survey on high-dimensional Gaussian process modeling with application to Bayesian optimization. *ACM Transactions on Evolutionary Learning and Optimization*, 2(2):1–26, 2022.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once for all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*, 2020.

Wuyang Chen, Xinyu Gong, and Zhangyang Wang. Neural architecture search on ImageNet in four GPU hours: A theoretically inspired perspective. In *International Conference on Learning Representations (ICLR)*, 2021.

Yu-Hsin Chen, Tushar Krishna, Joel S Emer, and Vivienne Sze. Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks. *IEEE Journal of Solid-State Circuits*, 52(1):127–138, 2016.

Hongrong Cheng, Miao Zhang, and Javen Qinfeng Shi. A survey on deep neural network pruning: Taxonomy, comparison, analysis, and recommendations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 46(12):10558–10578, 2024. doi: 10.1109/TPAMI.2024.3447085.

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for YouTube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, pp. 191–198, 2016.

Abhijit Das, Enrico Russo, and Maurizio Palesi. Multi-objective hardware-mapping co-optimisation for multi-DNN workloads on chiplet-based accelerators. *IEEE Transactions on Computers*, 2024.

Samuel Daulton, Maximilian Balandat, and Eytan Bakshy. Differentiable expected hypervolume improvement for parallel multi-objective Bayesian optimization. *Advances in Neural Information Processing Systems*, 33:9851–9864, 2020.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002. doi: 10.1109/4235.996017.

Radosvet Desislavov, Fernando Martínez-Plumed, and José Hernández-Orallo. Trends in AI inference energy consumption: Beyond the performance-vs-parameter laws of deep learning. *Sustainable Computing: Informatics and Systems*, 38:100857, 2023. ISSN 2210-5379. doi: https://doi.org/10.1016/j.suscom.2023.100857. URL https://www.sciencedirect.com/science/article/pii/S2210537923000124.

Xuanyi Dong, Lu Liu, Katarzyna Musial, and Bogdan Gabrys. NATS-Bench: Benchmarking NAS algorithms for architecture topology and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3634–3646, 2021.

Yawen Duan, Xin Chen, Hang Xu, Zewei Chen, Xiaodan Liang, Tong Zhang, and Zhenguo Li. TransNAS-Bench-101: Improving transferability and generalizability of cross-task neural architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5251–5260, 2021.

Łukasz Dudziak, Thomas Chau, Mohamed Abdelfattah, Royson Lee, Hyeji Kim, and Nicholas Lane. BRP-NAS: Prediction-based NAS using GCNs. *Advances in Neural Information Processing Systems*, 33:10480–10490, 2020.

Ziyi Guan, Wenyong Zhou, Yuan Ren, Rui Xie, Hao Yu, and Ngai Wong. A hardware-aware neural architecture search Pareto front exploration for in-memory computing. In *2022 IEEE 16th International Conference on Solid-State Integrated Circuit Technology (ICSICT)*, pp. 1–4, 2022. doi: 10.1109/ICSICT55466.2022.9963263.

Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. Neural collaborative filtering. In *Proceedings of the 26th International Conference on World Wide Web*, pp. 173–182, 2017.

Chi-Hung Hsu, Shu-Huan Chang, Da-Cheng Juan, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Shih-Chieh Chang. MONAS: Multi-objective neural architecture search using reinforcement learning. *CoRR*, 2018.

Weiwen Jiang, Lei Yang, Edwin Hsing-Mean Sha, Qingfeng Zhuge, Shouzhen Gu, Sakyasingha Dasgupta, Yiyu Shi, and Jingtong Hu. Hardware/software co-exploration of neural architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 39(12):4805–4815, 2020.

Weiwen Jiang, Qiuwen Lou, Zheyu Yan, Lei Yang, Jingtong Hu, Xiaobo Sharon Hu, and Yiyu Shi. Device-circuit-architecture co-exploration for computing-in-memory neural accelerators. *IEEE Transactions on Computers*, 70(4):595–605, 2021. doi: 10.1109/TC.2020.2991575.

Nikita Klyuchnikov, Ilya Trofimov, Ekaterina Artemova, Mikhail Salnikov, Maxim Fedorov, and Evgeny Burnaev. NAS-Bench-NLP: Neural architecture search benchmark for natural language processing. *IEEE Access*, 10:45736–45747, 2022. doi: 10.1109/ACCESS.2022.3169897.

Olga Krestinskaya, Mohammed E Fouda, Hadjer Benmeziane, Kaoutar El Maghraoui, Abu Sebastian, Wei D Lu, Mario Lanza, Hai Li, Fadi Kurdahi, Suhaib A Fahmy, et al. Neural architecture search for in-memory computing-based deep learning accelerators. *Nature Reviews Electrical Engineering*, pp. 1–17, 2024.

Olga Krestinskaya, Mohammed E. Fouda, Ahmed Eltawil, and Khaled N. Salama. Towards efficient IMC accelerator design through joint hardware-workload co-optimization. In *IEEE International Symposium on Circuits and Systems (ISCAS) 2025*, pp. 1–5, 2025.

Hyoukjun Kwon, Prasanth Chatarasi, Vivek Sarkar, Tushar Krishna, Michael Pellauer, and Angshuman Parashar. MAESTRO: A data-centric approach to understand reuse, performance, and hardware cost of DNN mappings. *IEEE Micro*, 40(3):20–29, 2020. doi: 10.1109/MM.2020.2985963.

Namhoon Lee, Thalaiyasingam Ajanthan, and Philip H. S. Torr. SNIP: Single-shot network pruning based on connection sensitivity. In *Proceedings of the 7th International Conference on Learning Representations*, 2019.

Cong Li, Yihan Yin, Xintong Wu, Jingchen Zhu, Zhutianya Gao, Dimin Niu, Qiang Wu, Xin Si, Yuan Xie, Guangyu Sun, and Chen Zhang. H^2-LLM: Hardware-dataflow co-exploration for heterogeneous hybrid-bonding-based low-batch LLM inference. In *Proceedings of the 52nd Annual International Symposium on Computer Architecture*, pp. 194–210, 2025. doi: 10.1145/3695053.3731008.

Dongrui Li, Tomomasa Yamasaki, Aarthy Mani, Anh Tuan Do, Niangjun Chen, and Bo Wang. LAXOR: A bit-accurate BNN accelerator with Latch-XOR logic for local computing. In *2023 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 1–6. IEEE, 2023a.

Dongrui Li, Ming Ming Wong, Yi Sheng Chong, Jun Zhou, Mohit Upadhyay, Ananta Balaji, Aarthy Mani, Weng Fai Wong, Li Shiuan Peh, Anh Tuan Do, et al. 1.63 pJ/SOP neuromorphic processor with integrated partial sum routers for in-network computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 32(11):2085–2092, 2024.

Guihong Li, Yuedong Yang, Kartikeya Bhardwaj, and Radu Marculescu. ZiCo: Zero-shot NAS via inverse coefficient of variation on gradients. In *The Eleventh International Conference on Learning Representations*, 2023b. URL https://openreview.net/forum?id=rwo-ls5GqGn.

Shang Liu, Wenji Fang, Yao Lu, Jing Wang, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. RTLCoder: Fully open-source and efficient LLM-assisted RTL code generation technique. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2024. doi: 10.1109/TCAD.2024.3483089.

Yuncheng Lu, Xin Zhang, Bo Wang, and Tony Tae-Hyoung Kim. SESOMP: A scalable and energy-efficient self-organizing map processor for IoT devices. *IEEE Journal of Solid-State Circuits*, 2024.

Joe Mellor, Jack Turner, Amos Storkey, and Elliot J Crowley. Neural architecture search without training. In Marina Meila and Tong Zhang (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 7588–7598. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/mellor21a.html.

Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pp. 1045–1048. Makuhari, 2010.

Maryam Parsa, Aayush Ankit, Amirkoushyar Ziabari, and Kaushik Roy. PABO: Pseudo agent-based multi-objective Bayesian hyperparameter optimization for efficient neural accelerator design. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2019. doi: 10.1109/ICCAD45719.2019.8942046.

Maryam Parsa, John P Mitchell, Catherine D Schuman, Robert M Patton, Thomas E Potok, and Kaushik Roy. Bayesian multi-objective hyperparameter optimization for accurate, fast, and efficient neural network accelerator design. *Frontiers in Neuroscience*, 14:667, 2020.

Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, Paul Whatmough, Matthew Mattina, and Tushar Krishna. A systematic methodology for characterizing scalability of DNN accelerators using SCALE-Sim. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 58–68, 2020. doi: 10.1109/ISPASS48437.2020.00016.

Lukas Sekanina. Neural architecture search and hardware accelerator co-search: A survey. *IEEE Access*, 9:151337–151362, 2021. doi: 10.1109/ACCESS.2021.3126685.

Rhea Sanjay Sukthanker, Arber Zela, Benedikt Staffler, Aaron Klein, Lennart Purucker, Jörg K. H. Franke, and Frank Hutter. HW-GPT-Bench: Hardware-aware architecture benchmark for language models. In *Advances in Neural Information Processing Systems*, volume 37, pp. 60776–60834, 2024.

Hanbo Sun, Zhenhua Zhu, Chenyu Wang, Xuefei Ning, Guohao Dai, Huazhong Yang, and Yu Wang. Gibbon: An efficient co-exploration framework of NN model and processing-in-memory architecture. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42 (11):4075–4089, 2023. doi: 10.1109/TCAD.2023.3262201.

Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. MnasNet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2820–2828, 2019.

Hidenori Tanaka, Daniel Kunin, Daniel L. Yamins, and Surya Ganguli. Pruning neural networks without any data by iteratively conserving synaptic flow. In *Advances in Neural Information Processing Systems*, volume 33, pp. 6377–6389, 2020.

Ye Tian, Langchun Si, Xingyi Zhang, Ran Cheng, Cheng He, Kay Chen Tan, and Yaochu Jin. Evolutionary large-scale multi-objective optimization: A survey. *ACM Computing Surveys (CSUR)*, 54(8):1–34, 2021.

Bo Wang, Manupa Karunarathne, Aditi Kulkarni, Tulika Mitra, and Li-Shiuan Peh. HyCUBE: A 0.9 V 26.4 MOPS/mW, 290 pJ/op, power efficient accelerator for IoT applications. In *2019 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, pp. 133–136, 2019. doi: 10.1109/A-SSCC47793.2019.9056954.

Tomomasa Yamasaki, Zhehui Wang, Tao Luo, Niangjun Chen, and Bo Wang. RBFleX-NAS: Training-free neural architecture search using radial basis function kernel and hyperparameter detection. *IEEE Transactions on Neural Networks and Learning Systems*, 36(6):10057–10071, 2025. doi: 10.1109/TNNLS.2025.3552693.

Jinda Yan, Yi Sheng, and Minghao Piao. Semantic segmentation-based wafer map mixed-type defect pattern recognition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(11):4065–4074, 2023. doi: 10.1109/TCAD.2023.3274958.

Amir R Zamir, Alexander Sax, William Shen, Leonidas J Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3712–3722, 2018.

Xin Zhang, Yuncheng Lu, Bo Wang, and Tony Tae-Hyoung Kim. A digital bit-reconfigurable versatile compute-in-memory macro for machine learning acceleration. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 70(5):1744–1748, 2023.

A Supplementary Analyses and Experimental Details

Additional motivation and background. Appendix Fig. 9 presents an illustrative cost breakdown for the iterative search stage, using 10 generated candidates per iteration over 30 iterations. Across MOBO, NSGA-II, and random search, training-based DNN evaluation accounts for 76.7% to 82.9% of the total search time. The remaining time includes hardware evaluation and the other search operations performed by each method. This calculation is separate from the experiment-specific initialization budgets reported in Section 3. The dominant evaluation cost motivates the use of training-free proxy scores.

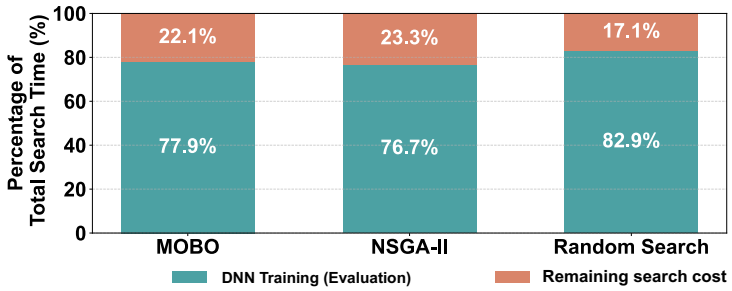


Figure 9: Illustrative breakdown of total search time into training-based DNN evaluation and the remaining search cost for MOBO, NSGA-II, and random search. The remaining component includes hardware evaluation and other search operations. The calculation covers 10 generated candidates per iteration over 30 iterations.

Training-free NAS reduces evaluation cost, but its proxy scores can differ from trained performance in scale and empirical distribution. As Appendix Fig. 10 illustrates, the distribution of mean intersection over union (*mIoU*) values obtained after 30 training epochs across all 4,096 architectures in TransNAS-Bench-101-Micro contains several well-separated modes. Training-free NAS methods instead compute scalar proxy scores on randomly initialized networks by aggregating architecture-sensitive statistics derived from activations or gradients. For instance, NASWOT (Mellor et al., 2021) encodes ReLU on/off patterns into binary activation codes, constructs a Hamming-distance kernel K_H over samples, and uses the log-determinant $\log|K_H|$ as its proxy score. The empirical proxy-score distributions produced by RBFleX-NAS, NASWOT, and ZiCO exhibit method-specific ranges and modal structures that differ from the post-training *mIoU* distribution over the same candidate architectures. This distributional comparison characterizes the available evaluation signals

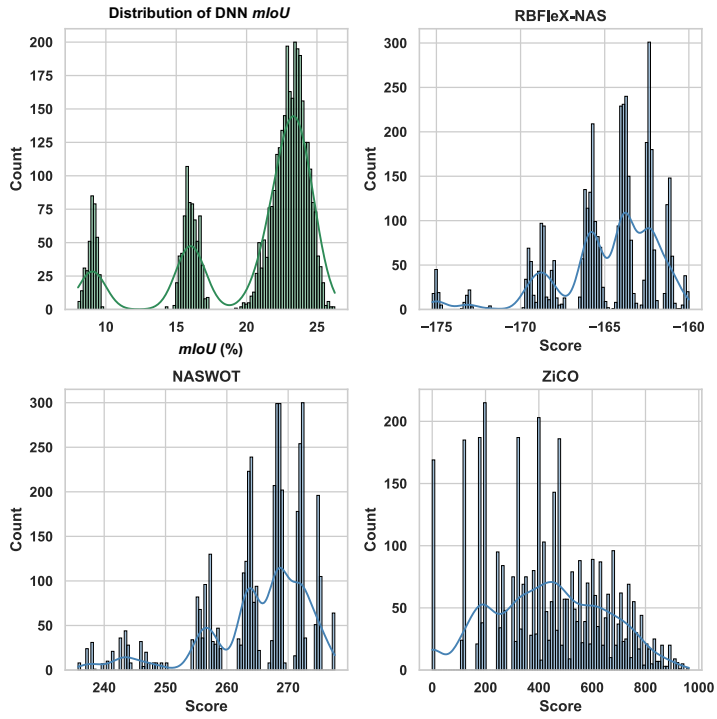


Figure 10: Distributions of ground-truth mean intersection over union (*mIoU*) values obtained after 30 training epochs and training-free proxy scores produced by RBFleX-NAS, NASWOT, and ZiCO across all 4,096 architectures in the TransNAS-Bench-101-Micro (Duan et al., 2021) semantic segmentation design space. The height of each histogram bin represents the number of architectures within the corresponding *mIoU* or score range, while the solid curves show the estimated probability density. Higher *mIoU* indicates higher segmentation performance, while higher proxy scores indicate higher predicted DNN performance under the corresponding training-free NAS method.

but cannot by itself establish whether these differences translate into differences in optimization performance.

To show how these evaluation signals enter DNN–hardware co-optimization, Appendix Fig. 11 illustrates a general Pareto optimization loop. Differentiable search (Guan et al., 2022), evolutionary algorithms (Sun et al., 2023), reinforcement learning (Jiang et al., 2021), and MOBO (Parsa et al., 2020; 2019) can be used to generate candidate configurations. The DNN candidates are evaluated using performance measured through training (Parsa et al., 2019; Guan et al., 2022; Jiang et al., 2020) or estimated by accuracy predictors (Dudziak et al., 2020), while the hardware configurations are evaluated using hardware efficiency predictors (Dudziak et al., 2020) or simulators (Krestinskaya et al., 2025). The resulting observations guide subsequent candidate generation, and the accumulated solutions are used to construct a Pareto front of non-dominated designs.

In conventional GP-based MOBO, each DNN–hardware candidate is represented by a joint input vector $\boldsymbol{X}_t$ that contains both neural architecture and hardware configuration variables. As the dimensionality of this joint input increases, accurately modeling the objective functions with GP surrogates becomes more difficult. After each evaluation, the observed configurations and their DNN performance (e.g., accuracy) and hardware performance metrics (e.g., energy and latency) are used to update the GP posteriors. The posterior means and uncertainties are then used by a multi-objective AF to score candidate configurations, and the next configuration $\boldsymbol{X}_{t+1}$ is selected by maximizing this AF.

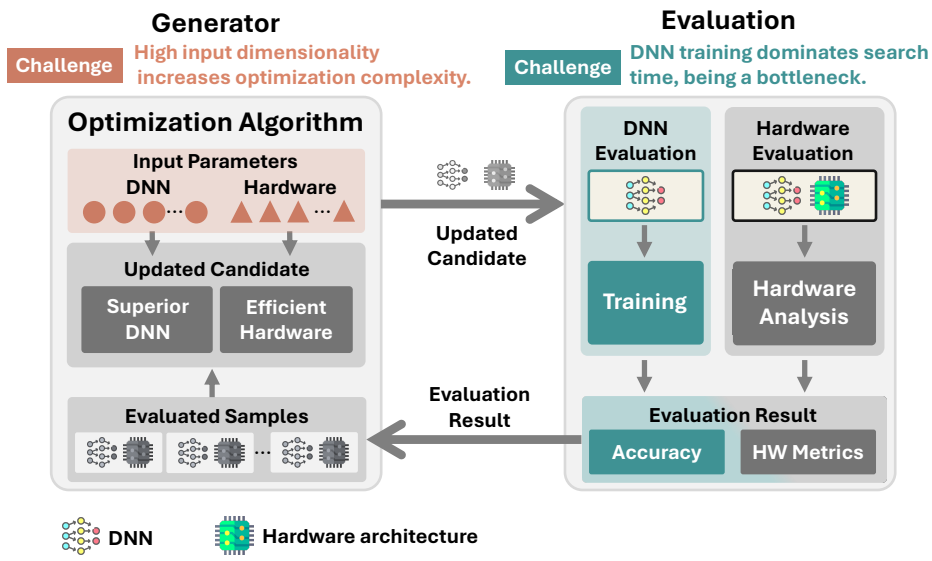


Figure 11: Illustration of Pareto-based multi-objective optimization for DNN and hardware configurations.

Additional method analysis and one-time preparation cost. Appendix Fig. 12 illustrates the resulting relationship between DNN ranks and RBFleX-NAS scores on TransNAS-Bench-101-Micro. This relationship is monotonic by construction because the candidates are ranked by sorting their proxy scores. The transformation does not alter the proxy-score values or remove multimodality from their underlying distribution. Instead, it replaces the original high-dimensional DNN encoding with a one-dimensional ordered input, allowing the surrogate to model the proxy score as a monotonic function of rank rather than as a function of the original high-dimensional DNN configuration. The resulting representation preserves the ordering induced by RBFleX-NAS.

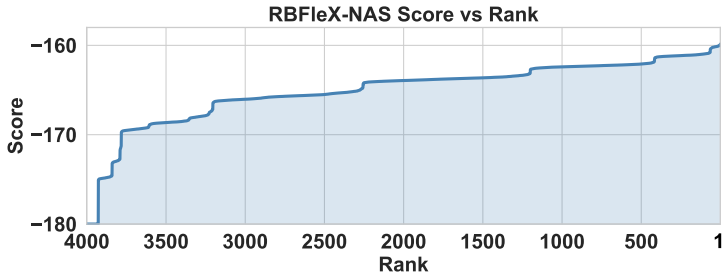


Figure 12: Monotonic relationship between DNN rank and RBFleX-NAS score on TransNAS-Bench-101-Micro for semantic segmentation.

Although the ranking table simplifies the co-optimization process, its construction incurs computational overhead. Specifically, generating the ranking table requires evaluating all candidate DNNs within the design space $\mathcal{D}_s$ using the training-free NAS proxy. To quantify this one-time preparation cost, we analyze its dependence on the number of candidate architectures. Here, let N denote the number of candidate architectures and B represent the mini-batch size, which is defined as the number of images used to evaluate each candidate network. As shown in RBFleX-NAS, the computational complexity for evaluating a single DNN architecture is $O(B^{2.373})$. The proxy-score evaluation term scales linearly with N. Evaluating all candidates therefore requires $O(NB^{2.373})$ time, while sorting the resulting proxy scores requires an additional $O(N \log N)$ time. Accordingly, the total computational cost C_{total} of constructing the ranking table is:

$$C_{total} = O\big(NB^{2.373} + N\log N\big)\,. \tag{7}$$

Because the ranking table contains one record for each candidate, its storage scales as $O(N)$ with respect to the number of candidates. During co-optimization, the precomputed proxy score of a candidate is retrieved through a table lookup, while its hardware metrics are obtained from the hardware evaluator. Measured runtime results that include this one-time preparation cost are reported for image classification in Section 3.1, for semantic segmentation in Appendix A under *Semantic segmentation setting and additional results*, and for the recurrent language modeling setting in Section 3.3.

We use SCALE-Sim (Samajdar et al., 2020) to evaluate the cycle counts of the systolic-array accelerator configurations considered in this work. As Appendix Fig. 13 shows, SCALE-Sim models systolic-array-based accelerator architectures and estimates execution cycle counts based on the specified hardware parameters and neural network configurations. Specifically, IFMAP, filters, and OFMAP are transferred from Dynamic Random-Access Memory (DRAM) to on-chip SRAM buffers through a memory controller. Tiled data are then streamed from SRAM to a two-dimensional processing element (PE) array, where each PE is equipped with local registers for filters (W_{reg}), activations (A_{reg}), and partial sums (P_{reg}), as well as a multiply-accumulate (MAC) unit.

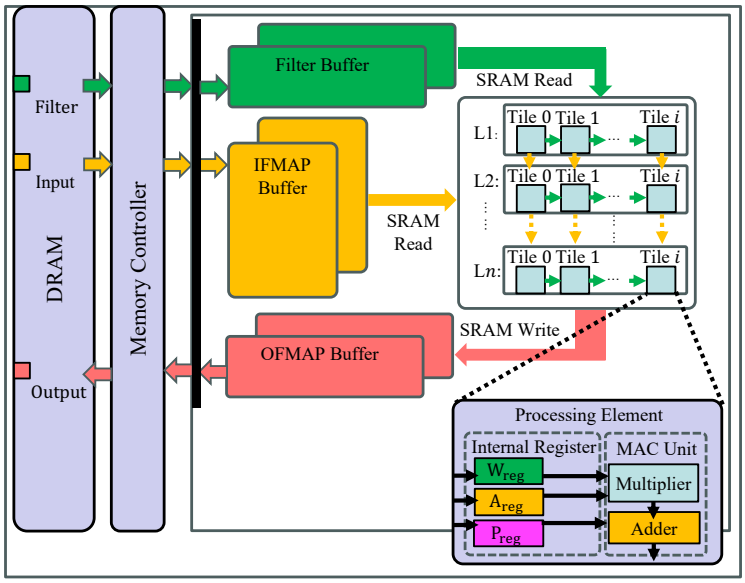


Figure 13: Illustration of overall accelerator architecture defined in SCALE-Sim.

Experimental protocols and design spaces. *Common MOBO implementation and reproducibility settings.*

All MOBO-based experiments use qNParEGO (Daulton et al., 2020) as the acquisition strategy. Candidate batches are constructed greedily using randomized augmented Chebyshev scalarizations, with a separate objective weight vector sampled for each candidate. For all evaluated settings, we implement qNParEGO in BoTorch (Balandat et al., 2020) using `qLogNoisyExpectedImprovement`. We use 128 Monte Carlo samples, 10 acquisition restarts, 512 raw samples, a batch limit of 5, and at most 200 acquisition-optimization iterations.

Each optimization objective is modeled using an independent `SingleTaskGP`, and the objective-specific models are combined using `ModelListGP`. Under BoTorch 0.10.0 and GPyTorch 1.11, the default covariance module is a scaled Matérn-$5/2$ kernel with automatic relevance determination over all input dimensions. The likelihood is a Gaussian likelihood with learned homoskedastic noise and a lower noise constraint of 10^{-4}. The surrogate models the selected DNN objective and negative cycle count in the image classification and semantic segmentation settings, and the selected

DNN objective and the negative analytical EDP proxy in the recurrent language modeling setting. For all evaluated methods in the image classification and semantic segmentation settings, the hypervolume values reported in this paper are recomputed using a common postprocessing protocol. For each evaluated pair of DNN and hardware configurations, the benchmark-provided task metric is converted to classification or segmentation error, and the corresponding SCALE-Sim cycle count is used as the hardware metric. This protocol is applied irrespective of the DNN objective used during optimization. After each batch is evaluated, the observations collected up to that iteration are standardized separately for each objective before the GP models are updated. All surrogate inputs are normalized to $[0, 1]$ using the fixed bounds of the corresponding design space. Acquisition optimization is performed in this continuous normalized space. The generated candidates are then mapped back to the original variable ranges using unnormalization followed by flooring and integer conversion. Discrete variables are represented by integer codes rather than one-hot encodings. Duplicate candidates are retained and counted as evaluation records.

Initial designs are generated using Sobol sampling. Each method uses independently generated initial designs drawn from the same design-space bounds, and the stored initial observations are reused for exact reproduction. The seeds used in each multi-seed evaluation are specified in the corresponding results. For the three settings presented in the main text, the reported co-optimization time covers GP fitting, acquisition optimization, and evaluation of newly proposed software and hardware candidates. Construction of the offline ranking table is reported separately as preparation time, while reported hypervolume postprocessing is excluded from the timing results. The total search time is calculated as the sum of the preparation and co-optimization times. The software environment uses Python 3.11.5, PyTorch 2.1.2 with CUDA 11.8 and cuDNN 8.7.0, BoTorch 0.10.0, GPyTorch 1.11, NumPy 1.26.4, SciPy 1.15.1, pandas 2.2.0, and NATS-Bench 1.8. The SCALE-Sim evaluations use the original implementation from the ARM-software repository at commit `bb76184f1f2b`.

Surrogate diagnostic. To examine how the rank representation affects GP prediction, we construct a diagnostic dataset containing 5,000 unique NATS-Bench-SSS architectures paired with 5,000 unique hardware configurations. Each record contains the corresponding RBFleX-NAS score and SCALE-Sim cycle count. For each of five data splits generated using seeds 42, 1234, 2026, 3407, and 777, 2,000 records are reserved for testing. Separate GPs are fitted using 100, 200, and 400 training records for each target and input formulation. The two targets are the RBFleX-NAS score and negative cycle count. The two input formulations are $(\mathbf{x}_s, \mathbf{x}_h)$ and $(r, \mathbf{x}_h)$, respectively. Both formulations use the same hardware variables. This protocol produces 60 GP fits. Appendix Table 2 reports the mean SRMSE and sample standard deviation across the five data splits. Lower SRMSE indicates lower prediction error. For each target, SRMSE is defined as

$$\mathrm{SRMSE} = \frac{\mathrm{RMSE}_{\mathrm{test}}}{s_{\mathrm{train}}},$$

where the root mean squared error ($\mathrm{RMSE}_{\mathrm{test}}$) is computed over the 2,000 held-out test records and s_{train} is the sample standard deviation of the corresponding training targets, calculated with $N_{\mathrm{train}} - 1$ degrees of freedom. Predictions and targets are transformed back to the original target scale before evaluation. The original DNN encoding and rank input use the same training and test records within each split and training size, resulting in the same normalization denominator for their paired comparison.

Table 2: GP surrogate diagnostic on NATS-Bench-SSS. Results are reported as mean standardized root mean squared error (SRMSE) $\pm$ sample standard deviation over five data splits. Lower values indicate lower prediction error.

Target	**Training size**	**Original DNN encoding**	**Rank input**
RBFleX-NAS score	100	0.1782 ± 0.0173	0.0899 ± 0.0290
RBFleX-NAS score	200	0.1354 ± 0.0149	0.0541 ± 0.0091
RBFleX-NAS score	400	0.1093 ± 0.0060	0.0367 ± 0.0086
Negative cycle count	100	2.0500 ± 1.5202	2.0504 ± 1.4727
Negative cycle count	200	2.0923 ± 1.4050	2.0262 ± 1.4813
Negative cycle count	400	1.4745 ± 0.6777	1.5490 ± 0.7068

The rank input achieves lower mean SRMSE for the RBFleX-NAS score at all three training sizes, reducing SRMSE by approximately 50%, 60%, and 66% with 100, 200, and 400 training records, respectively. It also yields lower SRMSE than the original DNN encoding in all five splits at each training size. Under this diagnostic protocol, the rank representation consistently improves GP prediction of the proxy score. For negative cycle count, the two input formulations achieve nearly identical mean SRMSE with 100 training records. The rank input achieves a slightly lower mean with 200 records, while the original DNN encoding achieves a lower mean with 400 records. Averaged across the 15 fits for each formulation, the corresponding mean SRMSE values are 1.87225 for the original DNN encoding and 1.87518 for the rank input, a difference of approximately 0.16%. Thus, the rank representation substantially improves modeling of the proxy score while maintaining comparable aggregate prediction error for negative cycle count. The cumulative hypervolume experiments separately evaluate how the rank formulation performs within the complete optimization process.

Compatibility with training-free NAS methods.

By default, our proposed OrdinalForge framework evaluates candidate DNNs using RBFleX-NAS. In this experiment, we assess the efficacy of OrdinalForge in co-optimization by replacing RBFleX-NAS with other training-free NAS algorithms, including NASWOT (Mellor et al., 2021), TE-NAS (Chen et al., 2021), snip (Lee et al., 2019), synflow (Tanaka et al., 2020), ZiCO (Li et al., 2023b), and grad_norm (Abdelfattah et al., 2021). We perform this investigation using the NAFBee (Yamasaki et al., 2025) benchmark, which explores activation functions over the CIFAR-10 dataset. Specifically, NAFBee provides the final accuracy of VGG-19 with 11 different activation functions, as summarized in Appendix Table 3. For the hardware design space, we employ SCALE-Sim to evaluate hardware metrics across various configurations, including the number of PEs, memory size, and dataflow options, as shown in Appendix Table 4.

Table 3: DNN design space on NAFBee (VGG-19) in terms of activation function

Component	**Design Options**
Activation	ReLU, GELU, SiLU, LeakyReLU, ReLU6, Mish, Hardswish, CELU, ELU, Hardtanh, SELU
Dataset	CIFAR-10

Table 4: Simulator-level SCALE-Sim hardware design space on NAFBee (VGG-19).

Component	**Design Options**
PE Array Height	1–64
PE Array Width	1–64
IFMAP SRAM Size	8–128 (KB)
Filter SRAM Size	8–128 (KB)
OFMAP SRAM Size	8–128 (KB)
Dataflow	Weight Stationary Output Stationary Input Stationary

To ensure a consistent basis for comparison, we keep the MOBO procedure unchanged across the evaluated training-free NAS methods. Each method constructs its own ranking table, and each proxy score is computed using a mini-batch of three images. For this first experiment, Bayesian optimization begins with 10 randomly selected initial designs and runs for 30 iterations with 10 newly generated candidates per iteration, resulting in 310 evaluated designs per method. In our experiments, each training-free NAS method assigns a proxy score $\rho(x)$ to a candidate architecture x. This score is used to construct the ranking table and guide optimization, whereas the post-training test error $f(x)$ obtained from NAFBee is used only for final evaluation. Since $\rho(x)$ and $f(x)$ differ in scale and optimization direction, $\rho(x)$ is treated as a ranking signal rather than as an additive noisy observation of $f(x)$. Random initialization and finite mini-batch sampling may introduce stochastic variation in proxy scores, but each score is computed once and then fixed throughout optimization.

Appendix Fig. 14 shows the co-optimization results, exhibiting a tradeoff between cycle count and DNN error rate. We use the test accuracy reported by NAFBee as the ground truth. Among the evaluated training-free NAS methods, RBFleX-NAS identifies a DNN–hardware configuration with the lowest observed CIFAR-10 error of 8.94% at 333,901 cycles. ZiCO, TE-NAS, and NASWOT reach the same hardware configuration but select DNNs with higher error.

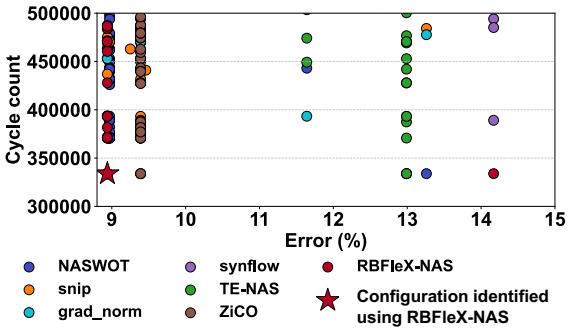


Figure 14: Optimization result on cycle count versus CIFAR-10 error rate (%). The red star marks the configuration identified through DNN and hardware co-optimization using RBFleX-NAS, achieving an error rate of 8.94% and a cycle count of 333,901.

Appendix Fig. 15 presents the hypervolume progression with respect to search cost across 30 iterations. The reference point to calculate hypervolumes is set to a classification error of 15% and a cycle count of 1.0×10^7. Hypervolume measures the volume of objective space dominated by the obtained Pareto front and bounded by the reference point. Larger hypervolume indicates a better Pareto set that simultaneously achieves lower classification error and fewer cycles. The co-optimization using RBFleX-NAS achieves the highest hypervolume of 5.86×10^7 with the lowest search cost of 1,128 seconds (i.e., 0.31 hours). In contrast, co-optimizations driven by ranking tables generated from NASWOT, snip, grad_norm, synflow, and TE-NAS fail to reach this hypervolume level. Although ZiCO attains a comparable hypervolume of 5.84×10^7, it requires $1.31\times$ longer search time, totaling 1482 seconds (i.e., 0.41 hours). These results support the selection of RBFleX-NAS as the default scoring algorithm in OrdinalForge. Within this compatibility study, its ranking table yields the highest hypervolume with the lowest measured search cost among the evaluated training-free scoring algorithms.

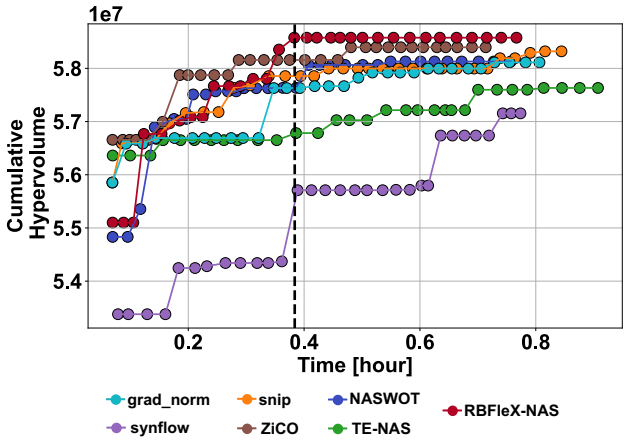


Figure 15: Hypervolume progression of training-free NAS methods over time. The reference point to calculate these hypervolumes is 15% error and 1.0×10^7 cycle count, respectively. A larger hypervolume indicates a better Pareto set covering a larger dominated region of the objective space. Time is measured for a single NVIDIA Tesla V100 GPU.

Appendix Table 5 summarizes OrdinalForge and the five baseline formulations using the metric names adopted in the image classification setting. In the semantic segmentation setting, *Accuracy metric* is replaced by *mIoU metric*. In the recurrent and causal language modeling settings, it is replaced by *Perplexity metric*. The remaining distinctions among the formulations are unchanged. The causal language modeling setting additionally includes MOBO using the NASWOT score with the original DNN encoding for the controlled representation comparison. Appendix Tables 6 and 7 define the DNN and hardware design spaces used for image classification. The same SCALE-Sim hardware design space is also used for semantic segmentation.

Table 5: Comparison of OrdinalForge and the five baseline formulations, shown using the image classification metrics

Method	Optimization Algorithm	DNN Evaluation Type	DNN Evaluation Timing	Training-Free Score	Ranking Table
(1) MOBO (Accuracy metric)	MOBO	Accuracy	Sequential	✗	✗
(2) Random Search (Accuracy metric)	Random Search	Accuracy	Sequential	✗	✗
(3) NSGA-II (Accuracy metric)	NSGA-II	Accuracy	Sequential	✗	✗
(4) MOBO (RBFleX-NAS score metric)	MOBO	RBFleX-NAS	Sequential	✓	✗
(5) MOBO (with a ranking table of NASWOT)	MOBO	NASWOT	Preparation	✓	✓
OrdinalForge (Ours)	MOBO	RBFleX-NAS	Preparation	✓	✓

Table 6: DNN design space on NATS-Bench-SSS in terms of channel size

Component	**Design Options**
1st Convolution Layer	8, 16, 24, 32, 40, 48, 56, 64
1st Cell Stage	8, 16, 24, 32, 40, 48, 56, 64
1st Residual Block	8, 16, 24, 32, 40, 48, 56, 64
2nd Cell Stage	8, 16, 24, 32, 40, 48, 56, 64
2nd Residual Block	8, 16, 24, 32, 40, 48, 56, 64
Dataset	ImageNet-16-120

Table 7: Simulator-level SCALE-Sim hardware design space used in the image classification and semantic segmentation settings.

Component	**Design Options**
PE Array Height	8–10,240
PE Array Width	8–10,240
IFMAP SRAM Size (KB)	8–20,480
Filter SRAM Size (KB)	8–20,480
OFMAP SRAM Size (KB)	8–20,480
Dataflow	Weight Stationary Output Stationary Input Stationary

The bounds in Appendix Table 7 define a common simulator-level design space for all evaluated methods. For the image classification and semantic segmentation settings, the hardware objective is the cycle count estimated by SCALE-Sim. Physical area and power are not included as additional objectives or constraints. These experiments compare co-optimization behavior within this shared simulated hardware space. As a boundary diagnostic, none of the seven configurations in the final non-dominated set of the OrdinalForge run using seed 42 places any of the five integer-valued hardware variables at its upper bound. Moreover, the configuration with the largest product of PE

array height and width does not achieve the lowest cycle count. Thus, this non-dominated set is not obtained by simply setting all hardware variables to their upper bounds.

Additional results for image classification. Fig. 3 illustrates the optimization results. Specifically, the software-hardware configuration identified by OrdinalForge achieves higher DNN accuracy and lower cycle count compared with the evaluated multi-objective optimization baselines.

Specifically, OrdinalForge identifies an accuracy-oriented configuration with a classification error of 52.9% and a SCALE-Sim-estimated cycle count of 150,096. It also identifies a delay-oriented configuration with an error rate of 63.5% and an estimated cycle count of 101,712. MOBO with a NASWOT-generated ranking table identifies the same accuracy-oriented DNN configuration as OrdinalForge. However, SCALE-Sim estimates at least 180 additional cycles for its selected hardware configuration relative to the configuration identified by OrdinalForge.

Appendix Fig. 16 compares the marginal distributions of final DNN accuracy and RBFleX-NAS proxy scores across the candidate architectures in NATS-Bench-SSS. The proxy score distribution contains two distinct modes, whereas the final accuracy distribution is concentrated around a single dominant mode. This distributional comparison characterizes the output ranges and modal structures of the two quantities, while the direct-score and rank-based formulations are compared separately through their optimization results.

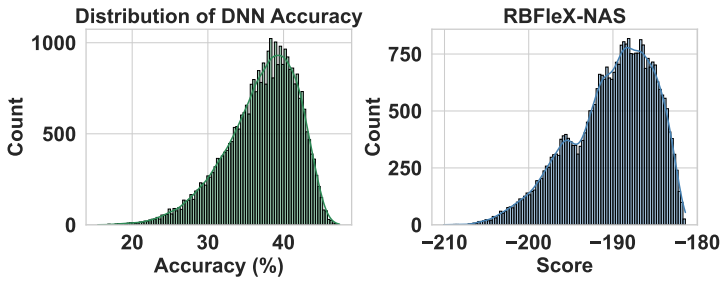


Figure 16: Marginal distributions of final DNN accuracy and RBFleX-NAS proxy scores across the candidate architectures in NATS-Bench-SSS. The proxy-score distribution contains two distinct modes, whereas the final-accuracy distribution contains one dominant mode.

Although NASWOT takes only 0.42 hours to generate a ranking table, its optimization phase takes 40.7 hours, resulting in a total search time of 41.1 hours, which is significantly longer than that of OrdinalForge. The observed difference in convergence behavior is consistent with the reported measurements of ranking quality: RBFleX-NAS achieves a Kendall correlation of 0.649 with the ground-truth DNN accuracies, compared with 0.402 for NASWOT. Moreover, the top-1 DNN selected by RBFleX-NAS achieves a lower error rate of 52.83% compared to 53.27% for NASWOT. Accuracy-based multi-objective optimization methods require significantly longer search time than OrdinalForge. Specifically, OrdinalForge achieves speedups of 5.1× over MOBO using the accuracy metric, 4.5× over random search, and 5.2× over NSGA-II.

Semantic segmentation setting and additional results. The semantic segmentation experiment uses the same five co-optimization baselines and optimization protocol as the image classification experiment in Section 3.1. The evaluated mIoU-based baselines are (1) MOBO, (2) random search, and (3) NSGA-II, all of which require DNN training (Krestinskaya et al., 2024). For further comparison, we additionally include two training-free approaches: (4) MOBO using RBFleX-NAS, which evaluates DNN candidates without training at each iteration, and (5) MOBO using a precomputed ranking table generated by NASWOT.

The DNN design space is defined by TransNAS-Bench-101-Micro (Duan et al., 2021), which contains 4,096 architectures evaluated on the Taskonomy dataset (Zamir et al., 2018). Appendix Table 8 illustrates the DNN design space defined in TransNAS-Bench-101-Micro. In this cell-level design space, each cell contains six operation positions (Operation #1–#6), and every position corresponds to a different edge in the cell Directed Acyclic Graph (DAG). All positions share the same candidate operator set (i.e., {Zeroize, Skip-connection, 1×1 conv, 3×3 conv}), but choosing operators at

Table 8: DNN design space in TransNAS-Bench-101-Micro

Component	Design Options
Operation #1	Zeroize, Skip-connection, 1×1 conv, 3×3 conv
Operation #2	Zeroize, Skip-connection, 1×1 conv, 3×3 conv
Operation #3	Zeroize, Skip-connection, 1×1 conv, 3×3 conv
Operation #4	Zeroize, Skip-connection, 1×1 conv, 3×3 conv
Operation #5	Zeroize, Skip-connection, 1×1 conv, 3×3 conv
Operation #6	Zeroize, Skip-connection, 1×1 conv, 3×3 conv
Dataset	Taskonomy

different positions leads to distinct architectures. These operations determine how the input feature map is transformed and propagated to subsequent operations. The semantic segmentation experiment uses the same simulator-level SCALE-Sim hardware design space and cycle-count evaluator as the image classification experiment in Section 3.1. Together with the TransNAS-Bench-101-Micro architecture space, the combined design space consists of 1.10×10^{25} possible configurations.

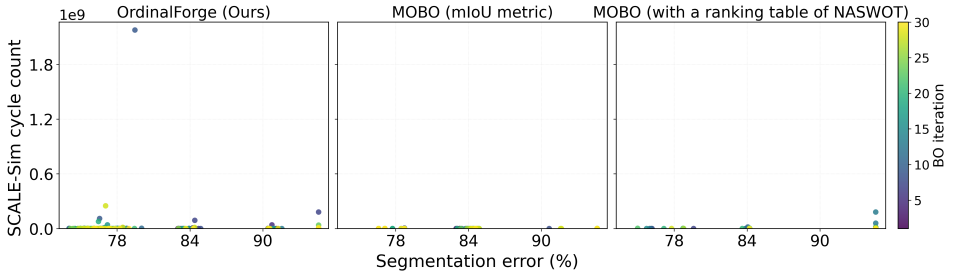
Fig. 5 shows the Pareto front of this experiment on the semantic segmentation task. OrdinalForge and MOBO with a NASWOT-based ranking table achieve less than 76% error, while NSGA-II and random search, which require DNN training, result in higher errors. OrdinalForge identifies a configuration with an error of 74.2% and a cycle count of 36,624, both lower than those of the configuration identified by MOBO with a NASWOT ranking table.

Fig. 4 presents the hypervolume trajectories from the reported runs. The hypervolume is computed with a reference point of 85% DNN error and 1.0×10^5 cycle count. OrdinalForge takes a total search time of 37.8 hours, including 0.13 hours for preparation and 37.7 hours for co-optimization, whereas MOBO with NASWOT takes 16.1 hours, including 0.16 hours for preparation and 15.9 hours for co-optimization. In the plotted run, OrdinalForge achieves a hypervolume of 6.83×10^5 after 15 iterations and identifies a configuration with an error of 74.2% and a cycle count of 36,624. OrdinalForge also exhibits a sharp hypervolume increase after 15 iterations in this trajectory, a trend not observed in the other plotted runs.

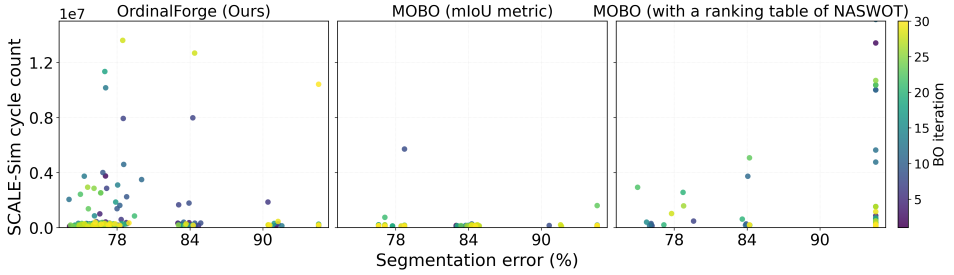
Appendix Fig. 17 visualizes all 300 candidates acquired by OrdinalForge and two MOBO baselines during 30 BO iterations. The 100 initialization samples are excluded, and point color indicates the corresponding BO iteration. In the enlarged region, the progression of point colors shows that OrdinalForge increasingly directs its search toward candidates with both lower segmentation error and lower cycle count, whereas the two MOBO baselines are concentrated at higher segmentation errors. The corresponding hypervolume comparison is reported in Appendix Fig. 4.

MOBO with a NASWOT-based ranking table uses the same number of input dimensions as OrdinalForge. The observed difference in candidate distributions is consistent with the ranking quality measurements reported in the RBFleX-NAS paper (Yamasaki et al., 2025). NASWOT and RBFleX-NAS achieve Kendall correlations of 0.339 and 0.381, respectively, on the TransNAS-Bench-101 micro design space for semantic segmentation. The top-ranked architectures identified by NASWOT and RBFleX-NAS have error rates of 78.13% and 74.67%, respectively, on the same benchmark.

In terms of optimization performance, this NASWOT-based MOBO variant reaches a hypervolume of 0.79×10^5 at the 30th iteration. Over the additional iterations represented by the dashed black segment in Appendix Fig. 4, its hypervolume remains below 4.0×10^5. MOBO using the RBFleX-NAS score metric requires less search time than OrdinalForge but reaches a lower hypervolume of 0.99×10^5. We further evaluate final hypervolume robustness in the semantic segmentation setting over three independent runs using seeds 42, 1234, and 2026. OrdinalForge achieves $6.21 \times 10^5 \pm 2.85 \times 10^5$ final cumulative hypervolume, with a CV of 45.89%. mIoU-based MOBO achieves $1.82 \times 10^5 \pm 1.13 \times 10^4$ with a CV of 6.22%. mIoU-based NSGA-II achieves $1.28 \times 10^5 \pm 1.81 \times 10^4$ with a CV of 14.09%. Direct score MOBO achieves $1.26 \times 10^5 \pm 3.14 \times 10^4$ with a CV of 24.83%, while MOBO using NASWOT ranking achieves $6.96 \times 10^4 \pm 9.31 \times 10^3$ with a CV of 13.38%. Although OrdinalForge exhibits greater variation across seeds in this setting, each of its three final hypervolume values exceeds every result obtained by the four baselines. Its mean final hypervolume is $3.41\times$ that of mIoU-based MOBO, $4.84\times$ that of mIoU-based NSGA-II, $4.91\times$ that of direct score



(a)



(b)

Figure 17: Candidate distributions during semantic segmentation optimization. Panel (a) shows all 300 candidates acquired by each method during the optimization stage, with 10 candidates per iteration over 30 BO iterations. The 100 initialization samples are excluded. Point color indicates the corresponding BO iteration. Panel (b) provides an enlarged view of panel (a) for SCALE-Sim cycle counts between 0 and 1.5×10^7.

MOBO, and $8.92\times$ that of MOBO using NASWOT ranking. The complete statistics are reported in Appendix Table 11. Similar to the results on NAS-Bench-SSS, Appendix Fig. 18 shows that the RBFleX-NAS score distribution exhibits a more pronounced multimodal structure than the ground-truth mIoU distribution. MOBO using the RBFleX-NAS score metric retains the six-dimensional DNN configuration as the surrogate input and achieves a lower hypervolume than OrdinalForge. OrdinalForge instead maps each DNN to its scalar score-based rank, allowing the surrogate to model the proxy score as a one-dimensional monotonic function of rank.

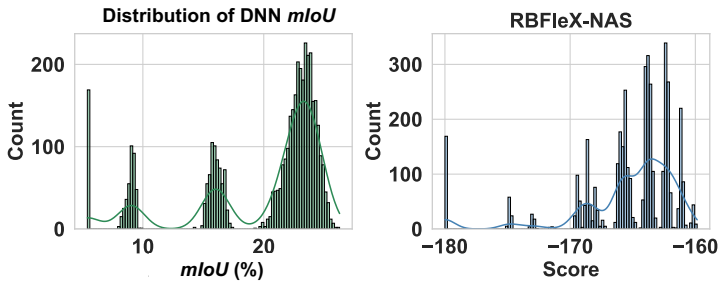


Figure 18: Distribution of DNN *mIoU* and RBFleX-NAS score on TransNAS-Bench-101-Micro (semantic segmentation). The RBFleX-NAS score distribution exhibits a more pronounced multimodal structure than the ground-truth *mIoU* distribution.

Additional results for the language modeling evaluations. *Recurrent language modeling setting and results.*

The image classification and semantic segmentation experiments use SCALE-Sim to estimate the cycle counts of CNN layers. NAS-Bench-NLP represents recurrent architectures through operation and connectivity choices rather than CNN layer topologies. Therefore, we use the analytical EDP proxy described in Section 3.3 for this setting. The proxy combines multiply-accumulate and memory-access terms for a shared embedding and decoder workload envelope as a function of four hardware parameters. All evaluated methods use the same proxy and parameter ranges, providing a consistent analytical EDP objective for the recurrent language modeling setting.

Appendix Table 9 summarizes the four hardware parameters used by the analytical EDP proxy. Together with the 14,322 NAS-Bench-NLP architectures, these parameter ranges define 9.88×10^{19} candidate combinations. For the recurrent language modeling setting, perplexity and the analytical EDP proxy are normalized using fixed division-based scaling:

$$\widetilde{\mathrm{PPL}}_i = \frac{\mathrm{PPL}_i}{97.7073158374454}, \qquad \widetilde{\mathrm{EDP}}_i = \frac{\mathrm{EDP}_i}{35.440037206037275}.$$

The scaling factors are fixed across all methods and random seeds. All methods and seeds use the common reference point

$$\mathbf{r} = (1.8,\ 2.5)$$

and treat $(\widetilde{\mathrm{PPL}}, \widetilde{\mathrm{EDP}})$ as a bi-objective minimization problem. For the candidate set $\mathcal{S}_t$ accumulated by checkpoint t, cumulative hypervolume is computed as

$$\mathrm{HV}_t = \lambda \left(\bigcup_{i \in \mathcal{S}_t,\ \widetilde{\mathbf{y}}_i \preceq \mathbf{r}} [\widetilde{\mathrm{PPL}}_i, 1.8] \times [\widetilde{\mathrm{EDP}}_i, 2.5] \right),$$

where $\widetilde{\mathbf{y}}_i = (\widetilde{\mathrm{PPL}}_i, \widetilde{\mathrm{EDP}}_i)$ and λ denotes two-dimensional area. Objective vectors outside the reference point do not contribute to the hypervolume. Invalid NAS-Bench-NLP models are assigned a perplexity sentinel of 10^{10} and therefore fall outside the reference point.

Table 9: Parameter space of the analytical EDP proxy used for the recurrent language modeling setting

Proxy parameter	**Design range**
Parallel MAC-capacity parameter	8–10,240
On-chip buffer-capacity parameter	8–20,480
Off-chip bandwidth parameter	8–20,480
Clock-frequency parameter	152–1,760

Appendix Fig. 7 presents the non-dominated solutions obtained by OrdinalForge and the baselines under the shared analytical proxy, illustrating the trade-off between normalized analytical EDP proxy values and normalized perplexity. Under this controlled comparison, OrdinalForge identifies favorable proxy value–perplexity trade-offs relative to NSGA-II, random search, and the MOBO variants. Although perplexity-based MOBO identifies a configuration with perplexity close to that of OrdinalForge, it has a substantially higher analytical EDP proxy value. OrdinalForge identifies two configurations, one with a normalized perplexity of 1.02 and a normalized EDP proxy value of 1.08, and another with a normalized perplexity of 1.05 and a normalized EDP proxy value of 1.01.

Although direct-score MOBO also employs RBFleX-NAS for candidate evaluation, it obtains lower hypervolume than OrdinalForge under the shared analytical EDP proxy. As Appendix Fig. 19 illustrates, the RBFleX-NAS score distribution exhibits a more pronounced multimodal structure than the ground-truth perplexity distribution. In this baseline, the surrogate models the proxy score as a function of the 78-dimensional DNN configuration, whereas OrdinalForge maps each DNN to its scalar score-based rank, allowing the surrogate to model the proxy score as a one-dimensional monotonic function of rank.

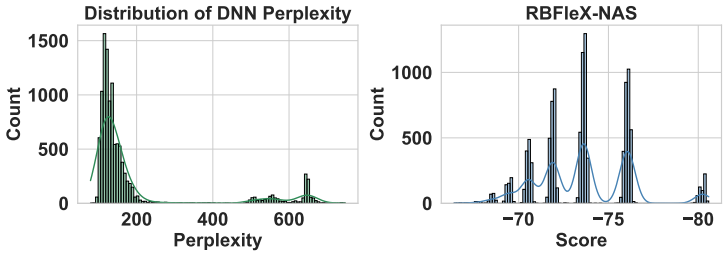


Figure 19: Distribution of DNN perplexity and RBFleX-NAS score on NAS-Bench-NLP. The RBFleX-NAS score distribution exhibits a more pronounced multimodal structure than the ground-truth perplexity distribution. Note that lower perplexity is better.

Fig. 6 shows the cumulative hypervolume achieved by each method after 30 iterations in the recurrent language modeling setting under the shared analytical EDP proxy. Hypervolume is computed with a reference point of 1.8 for normalized perplexity and 2.5 for the normalized analytical EDP proxy. In the runs summarized in the figure, OrdinalForge achieves the largest final cumulative hypervolume of 1.17 among the evaluated methods. We further repeat OrdinalForge and four baselines over five independent runs using seeds 42, 1234, 2026, 3407, and 777. OrdinalForge achieves 1.120 ± 0.106 final cumulative hypervolume, compared with 1.098 ± 0.061 for MOBO with NASWOT ranking, 0.834 ± 0.028 for perplexity-based MOBO, 0.686 ± 0.099 for perplexity-based NSGA-II, and 0.526 ± 0.021 for direct-score MOBO. The corresponding coefficients of variation are 9.48%, 5.60%, 3.39%, 14.48%, and 4.07%, respectively. OrdinalForge achieves a 2.03% higher mean than MOBO with NASWOT ranking, a 34.37% higher mean than perplexity-based MOBO, a 63.37% higher mean than perplexity-based NSGA-II, and a 113.05% higher mean than direct-score MOBO. All five OrdinalForge results exceed all five results from perplexity-based MOBO, perplexity-based NSGA-II, and direct-score MOBO. Together, these results show that OrdinalForge maintains the highest mean final cumulative hypervolume across random initializations under the shared evaluation protocol. The complete statistics are reported in Appendix Table 11.

Causal language modeling setting and results. We additionally evaluate the rank-based formulation on 10,000 valid GPT-S architectures from HW-GPT-Bench (Sukthanker et al., 2024). Appendix Table 10 summarizes the GPT-S architecture and H^2-LLM hardware design spaces used in this setting. The original DNN representation contains 27 variables: the number of Transformer layers, embedding dimension, bias option, 12 layerwise MLP ratios, and 12 layerwise attention-head counts. For architectures with fewer than 12 layers, inactive layerwise positions are padded with zeros. The hardware representation contains four variables and defines 32 executable configurations after applying the validity constraints of the fixed FFN-splitting mapping. The complete comparison domain contains 320,000 valid architecture–hardware pairs.

OrdinalForge replaces the 27-dimensional DNN input with the scalar rank induced by the RBFleX-NAS score, reducing the joint surrogate input from 31 dimensions to 5. The NASWOT ranking variant applies the same construction using NASWOT scores, allowing the rank representation to be evaluated with an alternative training-free proxy. In both ranking variants, rank is the surrogate input and the corresponding proxy score remains the software-side surrogate target. Both proxy tables are constructed once with seed 42 using the same fixed calibration batch of three OpenWebText sequences, the first 64 tokens of each sequence, and a shared randomly initialized GPT-S supernet, and are then reused across all search seeds.

HW-GPT-Bench provides the validation perplexity, and H^2-LLM (Li et al., 2025) estimates the analytical inference latency. For each GPT-S architecture, we construct the attention and feedforward

Table 10: GPT-S architecture and H^2-LLM hardware design spaces used in the causal language modeling setting. The experiment uses 10,000 valid GPT-S architectures sampled by HW-GPT-Bench from an architecture space containing approximately 10^{12} candidates.

GPT-S architecture design space		
Component	**Design Options**	**Scope**
Embedding dimension	192, 384, 768	Per architecture
Number of Transformer layers	10, 11, 12	Per architecture
Attention head count	4, 8, 12	Per active layer
MLP ratio	2, 3, 4	Per active layer
Bias	On, Off	Per architecture
H^2-LLM hardware design space		
Component	**Design Options**	**Scope**
NMP channel count	2, 4, 6	Per hardware configuration
FPU count per PE, PE frequency (MHz), PE bandwidth (GB/s)	$(8, 400, 6.4)$, $(8, 600, 6.4)$; $(8, 800, 6.4)$, $(8, 1000, 6.4)$; $(8, 400, 12.8)$, $(8, 600, 12.8)$; $(8, 800, 12.8)$, $(4, 1000, 12.8)$; $(8, 400, 25.6)$, $(8, 600, 25.6)$; $(4, 800, 25.6)$, $(4, 1000, 25.6)$; $(8, 400, 51.2)$, $(4, 600, 51.2)$; $(4, 800, 51.2)$, $(2, 1000, 51.2)$	16 coupled technology points

operator shapes for all active Transformer layers. H^2-LLM evaluates these operators with a batch size of 1, a prefill length of 1,024 tokens, a generation length of 128 tokens, a maximum context length of 2,048 tokens, and FP16 precision. The reported latency is the sum of the modeled operator latencies across all active layers. Components not represented by the H^2-LLM analytical model are not added separately. The resulting $10{,}000 \times 32$ latency matrix is computed once and then frozen for all methods. The fixed H^2-LLM configuration uses eight total channels, 16 banks per channel, 256 MB per bank, an FPU SIMD width of 16, 12.8 GB/s of NPU bandwidth per channel, and input, weight, and output buffers of 32, 32, and 64 KiB, respectively.

All methods use the same set of 10,000 GPT-S architectures, the same validation perplexity for each architecture, the same 32 hardware configurations, and the same frozen analytical latency value for each architecture–hardware pair. The RBFleX-NAS and NASWOT ranking tables order the architectures differently, but they do not change the underlying candidates or their evaluation values. Regardless of the proxy used during online optimization, the reported cumulative hypervolumes are recomputed in the common two-objective space of validation perplexity and analytical latency. During online search, both values are obtained by deterministic lookup rather than model training or a new H^2-LLM evaluation. Perplexity and latency are normalized using fixed global minimum and maximum values over the complete 320,000-pair domain, and hypervolume is computed with the reference point $(1.1, 1.1)$. All methods and search seeds share the same 100 canonical initial architecture–hardware pairs. Each run then evaluates 300 additional candidates, producing a total budget of 400 evaluations. We use search seeds 42, 1234, 2026, 3407, and 777. Because the initialization and offline evaluators are fixed, variation across these runs reflects the stochasticity of the online optimizers.

Multi-seed hypervolume summary.

Table 11: Final cumulative hypervolume across the three evaluation settings over independent runs. Results are reported as mean ± sample standard deviation. The CV is the sample standard deviation divided by the mean.

Evaluation setting	**Method**	**Runs**	**Final cumulative hypervolume Mean ± sample standard deviation**	**CV (%)**
Semantic segmentation	MOBO (RBFleX-NAS score, original DNN encoding)	3	$1.26 \times 10^5 \pm 3.14 \times 10^4$	24.83
	MOBO (NASWOT ranking)	3	$6.96 \times 10^4 \pm 9.31 \times 10^3$	13.38
	NSGA-II (mIoU metric)	3	$1.28 \times 10^5 \pm 1.81 \times 10^4$	14.09
	Random Search (mIoU metric)	3	$1.18 \times 10^5 \pm 3.22 \times 10^3$	2.72
	MOBO (mIoU metric)	3	$1.82 \times 10^5 \pm 1.13 \times 10^4$	6.22
	OrdinalForge (Ours)	3	$\mathbf{6.21 \times 10^5 \pm 2.85 \times 10^5}$	45.89
RNN-based language modeling (NAS-Bench-NLP)	MOBO (RBFleX-NAS score, original DNN encoding)	5	0.526 ± 0.021	4.07
	MOBO (NASWOT ranking)	5	1.098 ± 0.061	5.60
	NSGA-II (Perplexity metric)	5	0.686 ± 0.099	14.48
	Random Search (Perplexity metric)	5	0.958 ± 0.008	0.86
	MOBO (Perplexity metric)	5	0.834 ± 0.028	3.39
	OrdinalForge (Ours)	5	$\mathbf{1.120 \pm 0.106}$	9.48
Causal language modeling (HW-GPT-Bench)	MOBO (RBFleX-NAS score, original DNN encoding)	5	0.895 ± 0.006	0.70
	MOBO (NASWOT score, original DNN encoding)	5	0.887 ± 0.005	0.61
	MOBO (NASWOT ranking)	5	$\mathbf{0.916 \pm 0.002}$	**0.24**
	NSGA-II (Perplexity metric)	5	0.882 ± 0.004	0.48
	Random Search (Perplexity metric)	5	0.890 ± 0.009	1.05
	MOBO (Perplexity metric)	5	0.911 ± 0.005	0.57
	OrdinalForge (Ours)	5	0.914 ± 0.005	0.51