DyESP: Accelerating Hyperparameter-Architecture Search via Dynamic Exploration and Space Pruning

Xukun Liu[1], **Haoze Lv**[2], **Fenglong Ma**[3], **Chi Wang**[4], **Dongkuan (DK) Xu**[5]
[1]Northwestern University
[2]Southern University of Science and Technology
[3]The Pennsylvania State University
[4]Google DeepMind
[5]North Carolina State University
xukunliu2025@u.northwestern.edu, 11912814@mail.sustech.edu.cn, fenglong@psu.edu, wang.chi@microsoft.com, dxu27@ncsu.edu

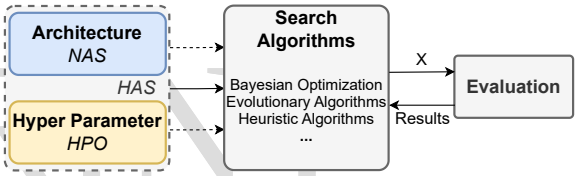
Abstract

In this work, we introduce DyESP, a novel approach that unites dynamic exploration with space pruning to expedite the combined search of hyperparameters and architecture, enhancing the efficiency and accuracy of hyperparameter-architecture search (HAS). Central to DyESP are two innovative components: a meta-scheduler that customizes the search strategy for varying spaces and a pruner designed to minimize the hyperparameter space by discarding suboptimal configurations. The meta-scheduler leverages historical data to dynamically refine the search direction, targeting the most promising areas while minimizing unnecessary exploration. Meanwhile, the pruner employs a surrogate model, specifically a fine-tuned multilayer perceptron (MLP), to predict and eliminate inferior configurations based on static metrics, thereby streamlining the search and conserving computational resources. The results from the pruner, which identifies and removes underperforming configurations, are fed into the meta-scheduler. This process updates the historical dataset used by the meta-scheduler, enabling it to adjust the exploration degree and refine the sampling strategy for subsequent iterations. This integration ensures the meta-scheduler is continually updated with relevant data, allowing for more accurate and timely adjustments to the exploration strategy. Experiments on various benchmarks show that DyESP outperforms existing methods in terms of both speed and stability on almost all benchmarks.

Code — https://anonymous.4open.science/r/DyESP-4B28

Introduction

Neural architecture search (NAS) (Zoph and Le 2017; Elsken, Metzen, and Hutter 2019; Zoph et al. 2018; Liu, Simonyan, and Yang 2019) and hyperparameter optimization (HPO) (Waring, Lindvall, and Umeton 2020; Feurer and Hutter 2019) are optimization techniques that automate the design of deep neural networks for downstream tasks. NAS aims to automatically design optimal neural network architectures for a given task. The input to NAS typically consists of a dataset and a defined task (e.g., image classification), while the output is an optimal or near-optimal neural network architecture customized for the task. Complementarily,



Wang,

Figure 1: Illustration of HAS pipeline.

HPO aims to find the best set of hyperparameters (settings or configurations) for a given model to maximize its performance on a specific task. The input to HPO is the range of possible hyperparameter values for a given model, and the output is the set of hyperparameter values that result in the best performance of the model, typically measured by accuracy, F1 score, or another relevant metric.

As shown in Fig. 1, joint hyperparameter and architecture search (HAS) aims to simultaneously optimize both the architecture of a neural network and its hyperparameters to find the best-performing model for a given task (Dong et al. 2020; Wang et al. 2022). The input to HAS typically includes the inputs of NAS and HPO, i.e., a dataset and a broad definition of possible neural network architectures and hyperparameter ranges. The output is the optimal neural network architecture along with its best set of hyperparameters, tailored to maximize performance on the specific task. HAS addresses the limitations of biased searching and unfair evaluation, prevalent in separate NAS and HPO approaches, by integrating the search spaces and evaluation processes. Consequently, HAS provides a more holistic and efficient exploration of the model configuration space, leading to better overall performance and more reliable comparisons between different models, and has been applied to various real-world tasks, such as medical image analysis (Prezja et al. 2023), tabular data modeling (Das and Dooley 2023), and deep learning workloads optimization (Nagrecha and Kumar 2023).

However, existing HAS approaches face the challenge of high computational costs, especially in the search part (Falkner, Klein, and Hutter 2018; Holland 2012; Hutter, Hoos, and Leyton-Brown 2011). The main reason for

the high computational cost is the vast and complex search space. Compared to NAS, which only searches the parameters of the model structure, the search space of HAS is not only larger due to the inclusion of the search for hyperparameters (Bansal et al. 2022; Hirose, Yoshinari, and Shirakawa 2021; Dai et al. 2021), but also contains a more complex mix of searches for discrete and continuous variables, which ultimately leads to an increase of one or even two orders of magnitude in the search space.

In this work, we propose a dynamic exploration and space pruning method, called DyESP, to accelerate the joint search of hyperparameter and architecture. DyESP mainly consists of two novel designs, the meta-learning-based scheduler (meta-scheduler) for accelerating search and a pruner for reducing the hyperparameter space. In particular, the meta-scheduler adapts the search strategy to different spaces, improving the efficiency of finding optimal configurations. It utilizes meta-learning principles to dynamically adjust the exploration degree, enabling a flexible and efficient search process. This unique approach allows the scheduler to learn from past searches, thereby optimizing the search strategy across various domains. The pruner is designed to accelerate HAS by effectively narrowing down the search space, and eliminating less promising configurations early on. It achieves this by utilizing a surrogate model approach, predicting the performance of different configurations based on static metrics, thus streamlining the evaluation process. This reduces the computational resources required for extensive evaluations, focusing efforts on more viable candidates. As a result, DyESP integrates the results from the pruner into the meta-scheduler through a structured, iterative process. After generating candidate configurations from the architectural and hyperparameter spaces, these are quickly evaluated for performance metrics such as the number of parameters and latency. The pruner then identifies and removes underperforming configurations. These actions culminate in updating the historical data set, which the meta-scheduler utilizes to adjust the exploration degree and refine the sampling strategy for subsequent iterations. This systematic integration ensures that the meta-scheduler operates with updated, pruned data, leading to a more focused and efficient search process.

Our main contributions can be summarized as follows:

- We propose a novel dynamic exploration strategy that leverages meta-learning. This strategy dynamically adjusts the search process based on learned patterns from previous searches, enhancing the efficiency of finding optimal solutions. The detailed design includes a meta-learning algorithm that continuously updates the exploration parameters, enabling the system to adapt its search strategy in real time to different search spaces and conditions, thus accelerating the hyperparameter and architecture search.
- We introduce a new space pruning technique that effectively reduces the search space by identifying and eliminating less promising paths in the search process. This approach significantly decreases the computational resources required. The detailed design involves a data-driven model that evaluates the potential of different search paths using historical data, which allows for a more focused and efficient search by concentrating on areas with higher potential for improvement. In addition, we propose the integration of a feedback loop between the meta-scheduler and the pruner, which enables the meta-scheduler to fine-tune its strategy based on the effectiveness of the pruning decisions, leading to a more synchronized and intelligent search.
- Extensive experiments across various benchmarks show the superior performance of DyESP over existing methods in terms of both efficiency and effectiveness, demonstrating its adaptability and robustness across different types of search spaces and optimization problems, highlighting the practicality and versatility in real-world scenarios. On NAS-Hpo-Bench-II (Hirose, Yoshinari, and Shirakawa 2021), DyESP achieves the optimal result with only 20% of the evaluation times required by the existing best-performing method. On NASBench-ASR(Mehrotra et al. 2021), NASBench-101(Ying et al. 2019), NASBench-Macro(Su et al. 2021), DyESP requires only 20%, 10.6%, 10% of the evaluation times of the optimal methods.

Methodology

DyESP is described in Fig. 2. It contains two components: meta-scheduler and pruner. The meta-scheduler dynamically adjusts the search range. The pruner, on the other hand, is employed to filter out poorly performing configurations before evaluation, thereby reducing the expensive cost of evaluation.

Preliminary

Stages of Search Algorithm. In the domain of NAS and HAS, the methodology of search algorithms is divided into two stages: global search and local search. Global search emphasizes a wide-ranging exploration aimed at identifying promising regions within the search space, and local search narrows the focus to a smaller area, meticulously optimizing configurations within this confined space. Existing research supports the efficacy of local search, particularly when advantageous initial points are predetermined (Dai et al. 2021).

Characteristics of Search Space. To explore the reasons behind these outcomes, we visualize multiple search spaces for both HAS and NAS, including areas such as computer vision (Ying et al. 2019) (Dong and Yang 2020), graphs (Qin et al. 2022), and other domains. The findings show: 1) The architectural landscape of HAS is a complex, multi-peaked domain, significantly marked by a high prevalence of local optima. 2) High-performing model architectures tend to cluster, suggesting a pattern of local regularity within the search space. These observations of global complexity and local regularity within the search spaces effectively underscore the reasons behind the superior performance of local search.

Exploration Degree. To quantitatively analyze the search types, we introduce the variable M as exploration de-

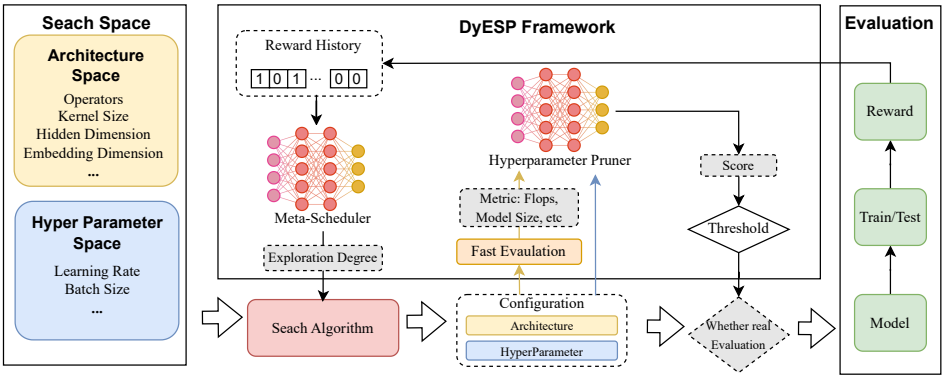


Figure 2: Overview of DyESP. DyESP incorporates a meta-controller and a hyperparameter pruner. First, the meta-scheduler guides the search algorithm in identifying promising configurations for models. Then, these configurations undergo a rapid evaluation. Finally, the most effective configurations are subjected to comprehensive evaluation through training and testing, which feeds back into the meta-scheduler to further improve the search strategy.

gree, which is defined in Eq. (1).

$$M = \frac{1}{dim} \sum_{i=1}^{dim} \frac{|\Delta D_i|}{R_i}, \tag{1}$$

where ΔD_i denotes the variation in the ith dimension, R_i is the range of values in the ith dimension, and dim represents the number of dimensions in the search space. Consequently, M serves as a quantifier of exploration degree throughout the search process, reflecting the extent to which the algorithm explores the search space. A high value of M suggests that the algorithm is engaging in global search, whereas a low value indicates a focus on local search.

Motivation: No Optimal Static Strategy! To maximize the effect of local search, the exploration degree should gradually decrease as the number of trials increases. Fig. 3 illustrates the relationship between different exploration degree trajectories and the final outcomes across various search spaces. Darker shades within the figure correlate with better results. Notably, NASBench101 appears to favor a balanced approach between exploration and exploitation, whereas NASBench201 shows poor performance under uniform distribution of these elements. This discrepancy clearly indicates that different search spaces have distinct preferences for search strategies, and consequently, a static search curve is not universally applicable. The findings underscore the necessity of an adaptive strategy, one that is capable of modulating its exploration degree in response to the evolving needs of the search space it operates within.

Meta-Scheduler: Enable efficient search in complex search space

To make the search algorithm adaptable to different search spaces, we propose a search strategy grounded in meta-learning, adaptable to any search space. This approach is designed to dynamically adjust the exploration degree, maintaining flexibility and efficiency in exploring through diverse search spaces.

Details of Meta-Scheduler. The Meta-Scheduler is a three-layer multilayer perceptron (MLP) structure to dynamically adjust the exploration of search algorithm. Specifically, it uses the history result H of the search algorithm shown in Eq. (2) as input to predict the exploratory degree for the current stage:

$$H_i = \begin{cases} 1 & \text{if } \mathcal{F}(\alpha_i) > \mathcal{F}(\alpha_{i-1}), \\ 0 & \text{otherwise,} \end{cases} \tag{2}$$

where $\mathcal{F}$ is the metric to be optimized such as accuracy, FLOPs, etc., α_i is the model configuration obtained from the i-th search, and H_i is the performance score from the i-th search. Meta-scheduler f_θ can be expressed as Eq. (3).

$$M_{i+1} = f_\theta(\{H_{i-k}, H_{i-k+1}, ..., H_i\}), \tag{3}$$

where k controls the length of history considered and M_{i+1} is the exploratory degree for the next search. By controlling the exploration degree based on historical results, the meta-scheduler can adapt the search strategy to different search spaces.

The optimization problem for search can be formulated as Eq. (4).

$$\alpha^* = \arg\max_{\alpha \in \Omega} \mathcal{F}\left(w^*(\alpha), \alpha\right), \tag{4}$$

where α represents the model configuration, which includes the architecture A and hyperparameter h, Ω denotes the search space, and w^* indicates the optimal weights for a specific configuration α. After introducing a meta-scheduler, the optimization problem is reformulated as Eq. (5).

$$\alpha^* = \arg\max_{\alpha \in \mathcal{P}(\Omega; f_\theta)} \mathcal{F}(w^*(\alpha), \alpha), \tag{5}$$

where f_θ denotes the meta-scheduler and $\mathcal{P}$ represents the configurations found in the search space Ω under the guidance of the search strategy implemented by the meta-scheduler f_θ.

Traing of Meta-Scheduler. To ensure that this scheduler improves performance across different search spaces, we employ meta-learning to train the scheduler, aiming to extract common knowledge from different spaces. The training process is described by Eq. (6).

$$\theta^* = \arg\min_{\theta} \sum_{\Omega_i \in \Omega_{support}} \mathcal{L}(\theta, \Omega_i), \tag{6}$$

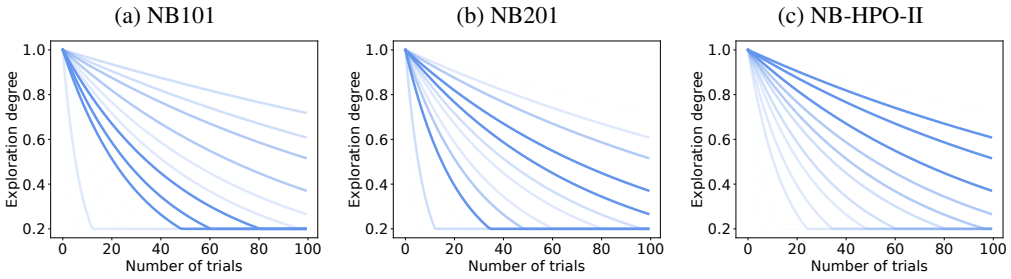


Figure 3: Best search curve in different spaces. Darker shades represent the curves that yield better final results. The result indicates that different search spaces have varying preferences for search types, and a fixed search curve can not be optimal for all spaces. Therefore, it is essential to have an adaptive strategy.

where θ^* represents the optimal parameters of the meta-scheduler, $\Omega_{support}$ represents the search spaces used for training the scheduler, and $\mathcal{L}(\theta, \Omega_i)$ denotes the loss incurred by the search algorithm when employing the search strategy $f(\theta)$ within the training search space Ω_i. To train this network, we define the loss function as Eq. (7).

$$\mathcal{L}(\theta, \Omega_i) = \sum_{k=1}^{K} \max(0, R(\theta, \alpha_i^k)) - R(\theta_i^{best}, \alpha_i^k) \cdot (\theta_i^{best} - \theta), \quad (7)$$

where K is the current number of search iterations, θ_i^{best} is the best search strategy currently known for the training search space Ω_i, and α_i^k is the configuration found in k-th search from the training search space Ω_i. The reward function $R(\theta, \alpha_i^k)$ measures the performance of the result obtained through searching under the k-th configuration α_i^k of the training search space Ω_i, given the current search strategy $f(\theta)$. Since the search curve is static and does not allow for dynamic adjustments based on historical information, simply emulating this curve is insufficient for achieving further improvements. Therefore, we dynamically update the value of θ_{best} during later stages of training when improvements are detected, actively encouraging the model to learn from θ_{best}.

Hyperparameter Pruner: Reduce vast search space

Considering the space complexity, training surrogate models to predict performance based on model architectures is costly. However, as neural network training typically involves a limited set of hyperparameters, constructing a unified surrogate model focusing on hyperparameter evaluation is more practical. Specifically, The pruner uses low-cost metrics, such as model size and FLOPs, obtained statically or as by-products during evaluation, along with the hyperparameters as inputs. The output of the pruner is a score for the input hyperparameter. This design ensures that the inputs to the pruner do not require costly training or extraction processes, thus not compromising the overall execution efficiency.

Due to the significant differences between search spaces, our pruner needs to be fine-tuned for specific search spaces during the search process. Notably, the pruner only seeks potential training data among the configurations that have been searched, thereby incurring no additional overhead from data collection. Moreover, due to the simplicity of the surrogate network used—a mere 3-layer MLP suffices—its training time is negligible compared to the costly evaluation overhead.

We train the pruner using a ranking loss and employ mean squared error (MSE) to ensure that the predicted values are not excessively divergent from the actual values. The loss function is given by Eq. (8).

$$\mathcal{L}(A, h_i, h_j) = \max(0, -\hat{y}_{ij} \cdot y_{ij}) + \lambda(\hat{y}_{ij} - y_{ij})^2, \quad (8)$$

where A represents the configuration's structure. h_i and h_j represent the two sets of hyperparameters to be compared. $\hat{y}_{ij}$ is the performance gap predicted by the pruner, y_{ij} is the actual performance gap, and λ is used to balance the proportion of the two types of loss.

Experiments

Experimental Setup

Benchmarks. We conduct experiments on several NAS and HAS benchmarks: NASBenchGraph (Qin et al. 2022), NAS-HPO-Bench-II (Hirose, Yoshinari, and Shirakawa 2021), JHAS-Bench-201 (Bansal et al. 2022), NASBench101 (Ying et al. 2019), NASBench201, NASBenchMacro (Su et al. 2021) and NASBenchASR (Mehrotra et al. 2021). These benchmarks are chosen to cover a broad range of neural network architectures and downstream tasks. Since these benchmarks pose different levels of challenges to NAS, we group them into 5 distinct difficulty levels, which are influenced by the search space and baseline performance, and specific values can be found in Table 1.

Level 1: The deviation from the optimal value is limited to 3%. **Level 2:** The deviation from the optimal value is limited to 2%, and slightly better than level 1. **Level 3:** The deviation from the optimal value is limited to 1%. **Level 4:** The deviation from the optimal value is limited to 0.5%. **Level 5:** The best result attainable by `DyESP` within 4000 trials.

Baselines. We include 6 representative search algorithms as baselines, including Random Search (RS) (Bergstra and Bengio 2012), Bayesian Optimization with hyperband (BOHB) (Falkner, Klein, and Hutter 2018), Reinforcement Learning (RL) (Zoph and Le 2017), Genetic Algorithm (GA) (Holland 2012), Differential Evolution (DE) (Storn and Price 1997), and Cost-Frugal Optimization (CFO) (Wu,

Benchmark	Level 1	Level 2	Level 3	Level 4	Level 5
NASBench101	92.77%	93.11%	93.37%	93.81%	94.18%
NASBench201	92.94%	93.30%	93.62%	94.17%	94.37%
NASBenchASR	87.00%	88.00%	89.00%	90.00%	90.50%
NASBenchMacro	92.80%	92.90%	93.00%	93.10%	93.12%
NASBenchGraph	80.00%	80.40%	80.60%	81.00%	81.60%
NATS	92.80%	92.90%	93.00%	93.10%	93.65%
Colorectal Histology[1]	94.80%	95.00%	95.50%	96.00%	96.20%
Fashion MNIST[1]	94.00%	94.30%	94.80%	95.00%	95.30%
Cifar-10[1]	90.50%	90.80%	91.00%	91.50%	91.80%

[1] Datasets from JHAS-Bench-201.

Table 1: Specific Values for Difficulty Levels.

Wang, and Huang 2021). To minimize randomness, all these algorithms are repeated 51 times using different random seeds. The median performance from these executions is selected for evaluation purposes.

Efficiency Comparison with Baselines

To probe the efficiency of DyESP, we evaluate the number of trials required to achieve different performance levels. For HAS tasks, Table 3 demonstrates the comparative outcomes of DyESP and baselines across various datasets within JHAS-Bench-201. DyESP leads the pack in almost all datasets, notably on the fashion MNIST dataset, where it surpasses the baseline results by a large margin, e.g., the baseline fails to reach the same level even after 5000 searches where the DyESP achieved with only 278 search trials. As depicted in Table 2, the advantage of DyESP is even more pronounced in NAS benchmarks. Specifically, on NASBench201, GA fails to discover a configuration as optimal as that found by DyESP within just 80 trials, despite undertaking 400 exploratory steps. Similarly, on NAS-HPO-Bench-II, BOHB requires nearly 10 times the resources to achieve a comparable outcome to that of DyESP. Experimental analyses across prominent benchmarks indicate that DyESP exhibits at least 2-fold improvement over all baselines within the search spaces explored.

These empirical results underscore the potential of DyESP across various benchmarks, owing to the implementation of the mete scheduler. As a result, DyESP is capable of exploring the search space with greater efficiency and precision compared to traditional algorithms that are dependent on heuristic or random searches. The validation of this superiority holds considerable significance for HAS tasks and other relevant domains. In the context of HAS tasks, DyESP offers an efficient and precise solution for the discovery of highly accurate models, thereby significantly reducing compute costs and enhancing search scalability.

Effect of the Proposed Pruner

We then verify the effectiveness of the proposed pruner on NAS-HPO-Bench-II. The results demonstrate that the pruner significantly reduces the number of search trials, thereby accelerating the search process. Specifically, compared to Random Search, BOHB, GA, and DE, our method achieves the highest acceleration, reducing the required trials from hundreds to only 41. Furthermore, integrating the pruner into our approach further decreases the trial count to 37, achieving an acceleration factor of 9.0× compared to Random Search.

Related Work

Neural Architecture Search (NAS). NAS was first proposed by (Kitano 1990) in 1990, and (Zoph and Le 2017) in 2016 made it popular again, which models the NAS problem as a black-box optimization problem (Elsken, Metzen, and Hutter 2019). For this problem, the quest to effectively traverse the vast search space has inspired researchers to employ a diverse assortment of strategies, such as evolutionary algorithms (Real et al. 2019), reinforcement learning approaches (Zoph et al. 2018), (Liu et al. 2018), and Bayesian optimization (Kandasamy et al. 2018), (White, Neiswanger, and Savani 2021). Since the above NAS methods usually need huge computing resources, researchers have again used several means to reduce the cost, such as gradient-based optimization (Liu, Simonyan, and Yang 2019), (Xu et al. 2020), (Fang et al. 2020), one-shot method, (Stamoulis et al. 2019), (Guo et al. 2020), etc. However, these approaches mainly focus on model architecture and neglect the important role of training hyperparameters, which can unintentionally distort the evaluation and introduce some bias.

Hyper-Parameter Optimization (HPO). HPO is a long-established machine learning technique for automatically finding the most appropriate parameter settings for a model, thereby improving its performance and efficiency (Waring, Lindvall, and Umeton 2020). Classical methods in this field include Bayesian optimization (Snoek, Larochelle, and Adams 2012), (Brochu, Cora, and de Freitas 2010), Random search (Bergstra and Bengio 2012), Gaussian process (Seeger 2004) gradient-based methods (Franceschi et al. 2017), etc. However, most methods used for HPO problems are designed to address issues within continuous spaces. When confronted with problems that combine both continuous and discrete spaces, these methods do not perform.

Joint Hyperparameter and Architecture Search (HAS). Early work in HAS can be traced back to 2016, (Mendoza et al. 2016) using the SMAC (Hutter, Hoos, and Leyton-Brown 2011) method to search for MLP structures and training hyperparameters. (Zela et al. 2018) highlighted the importance of HAS by demonstrating that the same hyperparameters do not apply to models with different structures. At this stage, HAS research is mainly divided into four directions: (1) *Improvement based on traditional HPO methods*: Research in this direction, including (Tiao et al. 2020) and (Zimmer, Lindauer, and Hutter 2021), focuses on exploring HAS search using BOHB (Falkner, Klein, and Hutter 2018) and its improvement.(Egele et al. 2021) using SMAC(Hutter, Hoos, and Leyton-Brown 2011) to find neural networks for solving Tabular Data. However, these methods based on Hyperparameter Optimization (HPO) often do not make specific improvements for the HAO search space, resulting in lower search efficiency compared to our algorithm. (2) *Using heuristics*: Approaches such as (Dai et al. 2021) combine traditional heuristics with agent models to search the complex HAS search space and achieve good performance on ImageNet. (3) *Using*

Benchmark	Task	Size	Method	Level 1	Level 2	Level 3	Level 4	Level 5	Acceleration
NASBench101	Vision	423k	Random Search	5	11	36	1100	4225	x1.0
			BOHB	3	5	25	380	914	x4.6
			GA	**4**	**4**	**17**	959	5000+	x0.8
			DE	**4**	**4**	25	261	2172	x1.9
			Ours	12	19	28	**140**	**432**	x9.8
NASBench201	Vision	15.6k	Random Search	15	26	729	5000+	5000+	x1.0
			BOHB	**5**	21	57	281	525	x9.5+
			GA	8	18	51	233	518	x9.7+
			DE	8	21	32	130	178	x28.1+
			Ours	10	**15**	**28**	**52**	**70**	x71.4+
NASBenchASR	ASR	8.2k	Random Search	132	198	214	424	5000+	x1.0
			BOHB	162	182	239	328	5000+	x1.0
			GA	440	604	4395	5000+	5000+	x1.0
			DE	63	85	**92**	**107**	5000+	x1.0
			Ours	**37**	**71**	104	**169**	**534**	x9.4+
NASBenchMacro	Vision	6.5k	Random Search	56	179	419	1851	2128	x1.0
			BOHB	44	61	94	261	425	x5.0
			GA	72	166	258	827	5000+	x0.4
			DE	32	42	56	63	5000+	x0.4
			Ours	**17**	**22**	**24**	**57**	**87**	x24.5
NASBenchGraph	Graph	26.2k	Random Search	21	76	137	588	2808	x2.0
			BOHB	24	83	107	1037	5000+	x0.6
			GA	35	404	463	5000+	5000+	x0.6
			DE	40	138	191	5000+	5000+	x0.6
			Ours	**12**	**23**	**40**	**137**	**1409**	x2.0
NATS	Vision	32.7k	Random Search	18	18	38	113	5000+	x1.0
			BOHB	35	40	61	134	5000+	x1.0
			GA	18	25	34	49	5000+	x1.0
			DE	21	23	**32**	46	150	x33.3+
			Ours	**10**	**10**	**10**	**11**	**124**	x40.3+

Table 2: Performance comparison of DyESP and baselines on NAS benchmarks. DyESP consistently outperforms baselines across all NAS benchmarks, achieving better results while reducing computational costs.

Dataset	Method	Level 1	Level 2	Level 3	Level 4	Level 5	Acceleration
cifar10	Random Search	1018	2817	5000+	5000+	5000+	x1.0
	BOHB	5000+	5000+	5000+	5000+	5000+	x1.0
	GA	989	1195	1584	2344	**3628**	x1.4+
	DE	5000+	5000+	5000+	5000+	5000+	x1.0
	Ours	**240**	**336**	**419**	**1293**	3701	x1.4+
fashion mnist	Random Search	**14**	25	146	465+	5000+	x1.0
	BOHB	445	21	804	5000+	5000+	x1.0
	GA	266	357	745	1264	5000+	x1.0
	DE	55	75	173	5000+	5000+	x1.0
	Ours	16	**20**	**34**	**53**	**278**	x18.0+
colorectal histology	Random Search	73	138	589	5000+	5000+	x1.0
	BOHB	2253	5000+	5000+	5000+	5000+	x1.0
	GA	724	989	2065	5000+	5000+	x1.0
	DE	5000+	5000+	5000+	5000+	5000+	x1.0
	Ours	**70**	**76**	**124**	**483**	**2093**	x2.4+

Table 3: Performance comparison of DyESP and baselines on JHAS-Bench-201. DyESP consistently outperforms traditional methods across all HAS benchmarks, achieving better results while reducing computational costs.

neural networks: (Dong et al. 2020) and (Wu et al. 2022) explore the HAS problem using neural networks like GCNs with some success. However, these neural network-based methods often lack transferability due to the differences in search spaces. (4) *Benchmark*: (Hirose, Yoshinari, and Shirakawa 2021) and (Bansal et al. 2022) have proposed HAS benchmarks, with the latter being the largest HAS benchmark to date. And (Klein and Hutter 2019) compares the performance of some classical HPO algorithms. As mentioned previously, existing HAS research mainly adapts HPO methods, yielding unsatisfactory performance and lacking generalization capabilities. Our work introduces a novel, generalized search method that controls the exploration degree and leverages prior search results for performance improvement.

Conclusions & Discussion

This paper introduces DyESP, a novel approach designed to expedite the joint hyperparameter and architecture search. At its core, DyESP employs a meta-learning-based scheduler alongside a versatile pruner to enhance search result quality while reducing search expenses. The scheduler adeptly navigates the search towards the most promising areas within the space, whereas the generic pruner streamlines this process by eliminating less promising hyperparameter candidates. A critical observation from our study is the identification of shared characteristics and variances between the NAS and HPO domains. This insight underscores the potential of meta-learning in refining traditional NAS techniques, presenting a valuable direction for future investigations.

Our experiments illustrate meta-learning's potential for dynamic search parameter tuning, However, the relatively simple structure of the meta-learning model used here can be further refined to enhance its performance. The design of such a meta-learning model for search algorithms presents a promising research direction (Hospedales et al. 2022). Additionally, we acknowledge the necessity of incorporating more information into our algorithm's generic pruner, which currently relies on limited data to determine candidates to retain or discard. By integrating additional information, such as the computation graph (Looks et al. 2017) of each candidate, the pruner could be more effective at shrinking the

search space and boosting the search process's efficiency.

Acknowledgments

This work was partially supported by the National Science Foundation (NSF) grants BCS-2416846 and OAC-2417850. The views and conclusions are those of the authors and should not be interpreted as representing the official policies of the funding agencies or the government.

References

Bansal, A.; Stoll, D.; Janowski, M.; Zela, A.; and Hutter, F. 2022. JAHS-Bench-201: A Foundation For Research On Joint Architecture And Hyperparameter Search. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Bergstra, J.; and Bengio, Y. 2012. Random Search for Hyper-Parameter Optimization. *J. Mach. Learn. Res.*, 13: 281–305.

Brochu, E.; Cora, V. M.; and de Freitas, N. 2010. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. *CoRR*, abs/1012.2599.

Dai, X.; Wan, A.; Zhang, P.; Wu, B.; He, Z.; Wei, Z.; Chen, K.; Tian, Y.; Yu, M.; Vajda, P.; and Gonzalez, J. E. 2021. FBNetV3: Joint Architecture-Recipe Search Using Predictor Pretraining. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR'2021, virtual, June 19-25, 2021*.

Das, R.; and Dooley, S. 2023. Fairer and More Accurate Tabular Models Through NAS. *arXiv preprint arXiv:2310.12145*.

Dong, X.; Tan, M.; Yu, A. W.; Peng, D.; Gabrys, B.; and Le, Q. V. 2020. AutoHAS: Differentiable Hyper-parameter and Architecture Search. *CoRR*, abs/2006.03656.

Dong, X.; and Yang, Y. 2020. NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search. *CoRR*, abs/2001.00326.

Egele, R.; Balaprakash, P.; Guyon, I.; Vishwanath, V.; Xia, F.; Stevens, R.; and Liu, Z. 2021. AgEBO-tabular: joint neural architecture and hyperparameter search with autotuned data-parallel training for tabular data. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 1–14.

Elsken, T.; Metzen, J. H.; and Hutter, F. 2019. Neural Architecture Search: A Survey. *J. Mach. Learn. Res.*, 20: 55:1–55:21.

Falkner, S.; Klein, A.; and Hutter, F. 2018. BOHB: Robust and Efficient Hyperparameter Optimization at Scale. In Dy, J. G.; and Krause, A., eds., *Proceedings of the 35th International Conference on Machine Learning, ICML'2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*.

Fang, J.; Sun, Y.; Zhang, Q.; Li, Y.; Liu, W.; and Wang, X. 2020. Densely Connected Search Space for More Flexible Neural Architecture Search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR'2020, Seattle, WA, USA, June 13-19, 2020*.

Feurer, M.; and Hutter, F. 2019. Hyperparameter Optimization. In Hutter, F.; Kotthoff, L.; and Vanschoren, J., eds., *Automated Machine Learning - Methods, Systems, Challenges*, The Springer Series on Challenges in Machine Learning, 3–33.

Franceschi, L.; Donini, M.; Frasconi, P.; and Pontil, M. 2017. Forward and Reverse Gradient-Based Hyperparameter Optimization. In Precup, D.; and Teh, Y. W., eds., *Proceedings of the 34th International Conference on Machine Learning, ICML'2017, Sydney, NSW, Australia, 6-11 August 2017*.

Guo, Z.; Zhang, X.; Mu, H.; Heng, W.; Liu, Z.; Wei, Y.; and Sun, J. 2020. Single Path One-Shot Neural Architecture Search with Uniform Sampling. In Vedaldi, A.; Bischof, H.; Brox, T.; and Frahm, J., eds., *Proceedings of the Computer Vision - ECCV'2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020*, volume 12361 of *Lecture Notes in Computer Science*, 544–560.

Hirose, Y.; Yoshinari, N.; and Shirakawa, S. 2021. NAS-HPO-Bench-II: A Benchmark Dataset on Joint Optimization of ConvolutionalNeural Network Architecture and Training Hyperparameters. In Balasubramanian, V. N.; and Tsang, I. W., eds., *Proceedings of the Asian Conference on Machine Learning, ACML'2021, 17-19 November 2021, Virtual Event,*.

Holland, J. H. 2012. Genetic algorithms. *Scholarpedia*, 7(12): 1482.

Hospedales, T.; Antoniou, A.; Micaelli, P.; and Storkey, A. 2022. Meta-Learning in Neural Networks: A Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(9): 5149–5169.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In Coello, C. A. C., ed., *Proceedings of the Learning and Intelligent Optimization - 5th International Conference, LION'5, Rome, Italy, January 17-21, 2011. Selected Papers*.

Kandasamy, K.; Neiswanger, W.; Schneider, J.; Póczos, B.; and Xing, E. P. 2018. Neural Architecture Search with Bayesian Optimisation and Optimal Transport. In Bengio, S.; Wallach, H. M.; Larochelle, H.; Grauman, K.; Cesa-Bianchi, N.; and Garnett, R., eds., *Proceedings of the 31st Annual Conference on Neural Information Processing Systems, NeurIPS'2018, December 3-8, 2018, Montréal, Canada*.

Kitano, H. 1990. Designing neural networks using genetic algorithms with graph generation system. *Complex System*, 4(4): 461–476.

Klein, A.; and Hutter, F. 2019. Tabular benchmarks for joint architecture and hyperparameter optimization. *arXiv preprint arXiv:1905.04970*.

Liu, C.; Zoph, B.; Neumann, M.; Shlens, J.; Hua, W.; Li, L.; Fei-Fei, L.; Yuille, A. L.; Huang, J.; and Murphy, K. 2018. Progressive Neural Architecture Search. In Ferrari, V.; Hebert, M.; Sminchisescu, C.; and Weiss, Y., eds., *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018*.

Liu, H.; Simonyan, K.; and Yang, Y. 2019. DARTS: Differentiable Architecture Search. In *Proceedings of the 7th International Conference on Learning Representations, ICLR'2019, New Orleans, LA, USA, May 6-9, 2019*.

Looks, M.; Herreshoff, M.; Hutchins, D.; and Norvig, P. 2017. Deep Learning with Dynamic Computation Graphs. arXiv:1702.02181.

Mehrotra, A.; Ramos, A. G. C. P.; Bhattacharya, S.; Dudziak, Ł.; Vipperla, R.; Chau, T.; Abdelfattah, M. S.; Ishtiaq, S.; and Lane, N. D. 2021. {NAS}-Bench-{ASR}: Reproducible Neural Architecture Search for Speech Recognition. In *International Conference on Learning Representations*.

Mendoza, H.; Klein, A.; Feurer, M.; Springenberg, J. T.; and Hutter, F. 2016. Towards Automatically-Tuned Neural Networks. In Hutter, F.; Kotthoff, L.; and Vanschoren, J., eds., *Proceedings of the 33rd International Conference on Machine Learning, ICML'2016, New York City, NY, USA*.

Nagrecha, K.; and Kumar, A. 2023. Saturn: An Optimized Data System for Multi-Large-Model Deep Learning Workloads (Information System Architectures).

Prezja, F.; Annala, L.; Kiiskinen, S.; Lahtinen, S.; and Ojala, T. 2023. Adaptive variance thresholding: A novel approach to improve existing deep transfer vision models and advance automatic knee-joint osteoarthritis classification. *arXiv preprint arXiv:2311.05799*.

Qin, Y.; Zhang, Z.; Wang, X.; Zhang, Z.; and Zhu, W. 2022. NAS-Bench-Graph: Benchmarking Graph Neural Architecture Search. arXiv:2206.09166.

Real, E.; Aggarwal, A.; Huang, Y.; and Le, Q. V. 2019. Regularized Evolution for Image Classifier Architecture Search. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence, AAAI'2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*.

Seeger, M. W. 2004. Gaussian Processes For Machine Learning. *Int. J. Neural Syst.*, 14(2): 69–106.

Snoek, J.; Larochelle, H.; and Adams, R. P. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In Bartlett, P. L.; Pereira, F. C. N.; Burges, C. J. C.; Bottou, L.; and Weinberger, K. Q., eds., *Proceedings of the 26th Annual Conference on Neural Information Processing Systems NeurIPS'2012, Lake Tahoe, Nevada, United States, December 3-6, 2012,*.

Stamoulis, D.; Ding, R.; Wang, D.; Lymberopoulos, D.; Priyantha, B.; Liu, J.; and Marculescu, D. 2019. Single-Path NAS: Designing Hardware-Efficient ConvNets in Less Than 4 Hours. In Brefeld, U.; Fromont, É.; Hotho, A.; Knobbe, A. J.; Maathuis, M. H.; and Robardet, C., eds., *Proceedings of the Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD'2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part II*.

Storn, R.; and Price, K. V. 1997. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11: 341–359.

Su, X.; Huang, T.; Li, Y.; You, S.; Wang, F.; Qian, C.; Zhang, C.; and Xu, C. 2021. Prioritized Architecture Sampling with Monto-Carlo Tree Search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10968–10977.

Tiao, L. C.; Klein, A.; Archambeau, C.; and Seeger, M. W. 2020. Model-based Asynchronous Hyperparameter Optimization. *CoRR*, abs/2003.10865.

Wang, Y.-Q.; Li, J.-Y.; Chen, C.-H.; Zhang, J.; and Zhan, Z.-H. 2022. Scale adaptive fitness evaluation-based particle swarm optimisation for hyperparameter and architecture optimisation in neural networks and deep learning. *CAAI Transactions on Intelligence Technology*.

Waring, J.; Lindvall, C.; and Umeton, R. 2020. Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artif. Intell. Medicine*, 104: 101822.

White, C.; Neiswanger, W.; and Savani, Y. 2021. BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search. In *Proceedings of the 35th Conference on Artificial Intelligence, AAAI'2021, Virtual Event, February 2-9, 2021*.

Wu, Q.; Wang, C.; and Huang, S. 2021. Frugal optimization for cost-related hyperparameters. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, 10347–10354.

Wu, X.; Zhang, D.; Zhang, M.; Guo, C.; Yang, B.; and Jensen, C. S. 2022. Joint Neural Architecture and Hyperparameter Search for Correlated Time Series Forecasting. *CoRR*, abs/2211.16126.

Xu, Y.; Xie, L.; Zhang, X.; Chen, X.; Qi, G.; Tian, Q.; and Xiong, H. 2020. PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search. In *Proceedings of the 8th International Conference on Learning Representations, ICLR'2020, Addis Ababa, Ethiopia, April 26-30, 2020*.

Ying, C.; Klein, A.; Real, E.; Christiansen, E.; Murphy, K.; and Hutter, F. 2019. NAS-Bench-101: Towards Reproducible Neural Architecture Search. *CoRR*, abs/1902.09635.

Zela, A.; Klein, A.; Falkner, S.; and Hutter, F. 2018. Towards Automated Deep Learning: Efficient Joint Neural Architecture and Hyperparameter Search. *CoRR*, abs/1807.06906.

Zimmer, L.; Lindauer, M.; and Hutter, F. 2021. Auto-Pytorch: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL. *IEEE Trans. Pattern Anal. Mach. Intell.*, 43(9): 3079–3090.

Zoph, B.; and Le, Q. V. 2017. Neural Architecture Search with Reinforcement Learning. In *Proceedings of the 5th International Conference on Learning Representations, ICLR'2017, Toulon, France, April 24-26, 2017*.

Zoph, B.; Vasudevan, V.; Shlens, J.; and Le, Q. V. 2018. Learning Transferable Architectures for Scalable Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR'2018, Salt Lake City, UT, USA, June 18-22, 2018*.