Information Dynamics and Applications
https://www.acadlore.com/journals/IDA

Novel Performance-Based Hyperparameter Optimization with the Use of Bounding Box Tuner (BBT)

Abdulvahap Mutlu, Şengül Doğan, Türker Tuncer

Department of Digital Forensics Engineering, Technology Faculty, Firat University, 23119 Elazig, Turkey

* Correspondence: Şengül Doğan (sdogan@firat.edu.tr)

Received: 06-01-2025 **Revised:** 06-15-2025 **Accepted:** 06-25-2025

Citation: A. Mutlu, Ş. Doğan, and T. Tuncer, "Novel performance-based hyperparameter optimization with the use of Bounding Box Tuner (BBT) ," *Inf. Dyn. Appl.*, vol. 4, no. 2, pp. 95–114, 2025. https://doi.org/10.56578/ida040204.

Abstract: Hyperparameter search was found not making good use of compute resources as surrogate-based optimizers consume extensive memory and demand long set-up time. Meanwhile, projects running with fixed budgets require lean tuning tools. The current study presents Bounding Box Tuner (BBT) and conducts tests of its capability to attain maximum validation accuracy while reducing tuning time and memory use. The project team compared BBT with Random Search, Gaussian Processes for Bayesian Optimization, Tree-Structured Parzen Estimator (TPE), Evolutionary Search and Local Search to decide on the optimum option. Modified National Institute of Standards and Technology (MNIST) classification with a multilayer perceptron (0.11 M weights) and Tiny Vision Transformer (TinyViT) (9.5 M weights) were adopted. Each optimizer was assigned to run 50 trials. During the trial, early pruning stopped a run if validation loss rose for four epochs. All tests applied one NVIDIA GTX 1650 Ti GPU; the key metrics for measurement included best validation accuracy, total search time, and time per trial. As regards the perceptron task, BBT reached 97.88% validation accuracy in 1994 s whereas TPE obtained 97.98% in 2976 s. Concerning TinyViT, BBT achieved 94.92% in 2364 s, and GP-Bayesian gained 94.66% in 2191 s. It was discovered that BBT kept accuracy within 0.1 percentage points of the best competitor and reduced tuning time by one-third. The algorithm renders the surrogate model unnecessary, enforces constraints by design and exposes solely three user parameters. Supported by the evidence of these benefits, BBT was considered to be a practical option for rapid and resource-aware hyperparameter optimization in deep-learning pipelines.

Keywords: Bounding Box Tuner; Hyperparameter optimization; Performance-based tuning; Hyperparameter optimization methods; Axis-aligned sampling; Deep learning; Machine learning

1 Introduction

Hyperparameter optimization (HPO) constitutes the basis of modern machine learning workflows. Even apparently modest neural networks expose an intertwined set of learning rates, architectural depths, regularization coefficients and batch sizes whose combined values determine training stability, convergence speed and most importantly, generalization. Over the past decade, practitioners have migrated from exhaustive grid search to more sophisticated strategies such as random sampling, Bayesian Optimization with Gaussian Processes or Tree-Structured Parzen Estimators, successive-halving schemes (e.g., Hyperband and Asynchronous Successive Halving Algorithm (ASHA)), as well as population-based evolutionary algorithms [1–6]. These methods demonstrate their capability in reducing the sessions of full-length training required to locate near-optimal configurations, yet each carries trade-offs that affect their performance in time- or resource- constrained environments [7].

Bayesian optimization achieves data-efficient exploration through surrogate modelling but incurs expensive model retraining cost for iteration, scales unsatisfactorily as the dimensionality or noise of the search space elevates, and frequently ignores meaningful inter-parameter interactions by relying on separable kernels or univariate acquisition updates.

Successive-halving families exploit early stopping to cull weak configurations; however, they treat the admissible hyperparameter domain as static, hence devoting budget to regions that soon prove unpromising.

Evolutionary and population-based training embrace non-convex landscapes by mutating and recombining entire parameter vectors. Nevertheless, they generally require hundreds of objective evaluations before convergence; unguarded mutation could also result in excessive budget expenditure on implausible regions [8, 9] .

As a consequence, researchers and engineers deploying deep models in edge devices, research prototypes or limited-credit cloud platforms still grapple with long tuning cycles and escalating costs, particularly when they cannot leverage large, distributed clusters or proprietary Automated Machine Learning (AutoML) systems [10, 11].

1.1 Literature Review

1.1.1 Random and grid search

Exhaustive grid search was one of the earliest strategies for hyperparameter tuning, systematically evaluating points on a predefined grid in the hyperparameter space. Despite being straightforward, grid search suffers from the curse of dimensionality: the number of combinations grows exponentially along with the number of hyperparameters, hence curtailing the feasibility to cover a fine-grained grid when more than a few hyperparameters are involved.

Random search offers a straightforward and effective alternative by sampling hyperparameter configurations uniformly at random from the search space. Contrary to expectations, random search has been demonstrated to outperform grid search in high-dimensional settings because, with the same number of trials, it explores a broader range of values for each discrete hyperparameter rather than redundantly evaluating many combinations along a rigid and systematic grid. Random search is completely exploration-oriented; despite its comprehensive exploration of the hyperparameter landscape, it does not incorporate any learning from past trials to guide future searches. Yet, the search itself could serve as a robust baseline in numerous studies due to its simplicity and ability to scale with arbitrary dimensions and search spaces. In practice, random search is frequently used in combination with early stopping heuristics or as a component within more advanced algorithms (e.g., as the initial exploration phase).

1.1.2 Bayesian optimization

Bayesian optimization is a model-based approach to hyperparameter tuning that has gained prominence for its efficiency in searching expensive, black-box functions [12]. The essential idea is to establish a probabilistic surrogate model of the objective function (e.g., validation error as a function of hyperparameters) based on the results of previous trials, and then apply this model to plan for the setting of the next hyperparameter. Typically, Bayesian HPO methods apply a Gaussian Process (GP) or a Tree-Structured Parzen Estimator (TPE) or other regression model to the observed performance data [13]. This surrogate provides a predictive distribution for the performance of any candidate hyperparameter configuration, which is then used by an acquisition function to balance exploration and exploitation when proposing the next query point.

Bayesian optimization has been successfully utilized to tuning machine learning algorithms, usually achieving better results with far fewer evaluations than random or grid search, especially when the hyperparameter search space ranges from low- to moderate-dimensional and the cost of training/evaluating one configuration is substantial. The approach explicitly manages the exploration-exploitation trade-off: early iterations will explore broadly since uncertainty in the surrogate is high everywhere. As data accumulate, the model homes in on promising regions, i.e., exploiting the surrogate's knowledge to fine-tune the best observed peaks [14]. Theoretically speaking, under certain smoothness assumptions and appropriate acquisition functions, Bayesian optimization is found to converge towards the global optimum, with bounds on regret (difference occurs between the found solution and the true optimum) that decay as more evaluations have been performed [15].

1.1.3 Evolutionary and population-based algorithms

Evolutionary algorithms approach HPO by mimicking principles of natural evolution, searching for good configurations through a process of mutation, recombination, and selection applied to a population of candidate solutions. In the context of HPO, an individual in the population encodes a set of hyperparameters as a "chromosome" string. A genetic algorithm (GA) will randomly initialize a population of such individuals in hyperparameter settings, evaluate their fitness (e.g., the negative validation error of a model trained with those hyperparameters), and then iteratively produce new generations by selecting fitter individuals to breed/crossover and randomly mutating some of their hyperparameter values. Over successive generations, the population tends to accumulate better hyperparameter configurations, as high-performing regions of the search space are exploited and combined, while mutation introduces new exploratory variations. Other evolutionary and bio-inspired algorithms like Particle Swarm Optimization (PSO) and Differential Evolution (DE) have been applied to hyperparameter search; these maintain a set of candidate solutions that move through the search space influenced by their own and their neighbors' past successes (PSO) or by differential mutations of population members (DE), gradually converging toward optima [16, 17]. Evolutionary approaches are particularly flexible as exemplified by their handling of discrete and continuous hyperparameters, complex mixed-type spaces, and multiple objectives naturally [18]. They typically do not rely on smoothness or differentiability of the objective, but on the capability to evaluate hyperparameter configurations. This renders them more robust at the cost of frequently requiring a considerable number of evaluations to reach convergence, especially if the population size and the number of generations are large. In practice, evolutionary algorithms have been utilized effectively for tuning deep networks and in neural architecture search, where the search space is significantly large

and highly non-convex; they have discovered novel architectures or hyperparameter schedules that human experts might not consider beforehand [19, 20].

A notable family of evolutionary HPO methods in deep learning is population-based training (PBT) [8]. PBT is a specific strategy that blends evolutionary ideas with online training of models: instead of evaluating each hyperparameter configuration from scratch, PBT maintains a population of neural network training processes running in parallel, each with its own hyperparameters. At periodic intervals (epochs or batches), the algorithm selects the best-performing models and exploits them by replacing the worst-performing models with copies of the best (including their learned weights), then explores by mutating the hyperparameters of those copied models (e.g., slightly perturbing the learning rate or dropout rate). This way, PBT effectively tunes hyperparameters on the fly as part of the training process, continually adapting hyperparameters based on intermediate results. Over time, less effective hyperparameter settings are pruned out, and all training resources are shifted toward variants of the better settings.

1.1.4 Local search and greedy strategies

Local Search and greedy strategies focus on iteratively improving a single hyperparameter configuration by making local changes, rather than maintaining a global population or probabilistic model of the entire space. These methods start from an initial guess (which could be randomly chosen or informed by prior knowledge) and then proceed to greedily adjust hyperparameters in order to reduce the objective (e.g., validation loss), one small step at a time. A simple example is coordinate descent, an optimization algorithm that reduces a function by optimizing along coordinate directions iteratively, on hyperparameters. In other words, one can fix all but one hyperparameter and then do a one-dimensional search (or just experiment with a few increments/decrements) to locate a better value for that hyperparameter, then proceed to another hyperparameter, cycling through them iteratively. This technique treats each hyperparameter separately and ignores interactions in the short term, but can be effective if the hyperparameters are roughly independent or if a good solution can be found by sequentially tuning each to optimality. Another approach is iterative refinement or hill climbing, where at each step, the algorithm tries small random perturbations of the current best hyperparameter set (e.g., slightly increasing or decreasing a continuous hyperparameter, or toggling a categorical option) and moves to the new configuration if it improves performance [21]. After numerous iterations, hill climbing will pursue a path of incremental improvements until it reaches a point where no local tweak yields a satisfactory consequence (a local optimum). Greedy Local Searches are typically fast and simple to implement; they may operate effectively when the hyperparameter space is smooth and unimodal around the optimum so that a local optimum is likely the global optimum. Nevertheless, in deep learning, hyperparameter landscapes are often rugged and multimodal, namely a purely local method could become entangle in a suboptimal region because it fails to take into account more radical changes that would adversely affect performance at first but later lead to a better configuration [22, 23]. Despite their limitations, Local Search strategies could play a role in HPO [24].

1.1.5 Multi-fidelity and hybrid approaches

A significant development in hyperparameter optimization, especially pertinent to deep learning, is the class of multi-fidelity methods [25, 26]. These methods recognize that one can get partial information about the quality of a hyperparameter configuration by running a truncated or smaller version of the training job, thereby saving time and resources compared to running full training for every trial all the time. Evaluating a neural network after only a few epochs (or on a subset of the data, or with a smaller model size) provides a noisy estimate of its final performance, but this estimate is valuable to decide if further training should be conducted [25]. Multi-fidelity HPO algorithms implement this concept to allocate varying amounts of resources to different trials, pruning the unpromising ones early and providing full training selectively to the most promising configurations [27].

A basic algorithm in this category is the Successive Halving Algorithm (SHA) [4]. SHA begins by training a vast number of candidate hyperparameter configurations for a short-term budget, for instance, a few epochs each. After this initial stage, only the top-performing fraction of configurations, say the top 50%, are selected to continue the next step when the remaining ones are discarded. The "survivors" are then trained for a long-term budget with more epochs, and the selection process repeats to retain the top fraction, i.e., the best performers. This halving process continues several rounds until the final round, when a small number of, possibly one, configuration(s) are trained for the maximum budget. SHA is designed to spend most of its computation on the more promising hyperparameters, while quickly eliminating substandard performers to gather resources. The "racing" mechanism of SHA ensures a form of exploitation with focuses on winners after the initial exploration, effectively after a certain number of short trials. Theoretical analysis demonstrates that SHA could promptly locate the existence of an authentically optimal hyperparameter configuration, within a logarithmic factor, and its performance is comparable to an idealized oracle with the foresight to allocate resources [28]. Nevertheless, the drawback of SHA was its dependence on the choice of initial budget and the downsampling rate, namely the fraction of data to be kept in each round, so as to guarantee its effectiveness. If the initial budget is either meagre or enormous relative to the problem to be resolved, SHA might prematurely discard good configuration or waste resources, respectively.

Hyperband is an extension of successive halving that addresses the problem of determining the appropriate budget a priori. Hyperband essentially runs multiple trials of SHA with different settings in parallel or interleaved: all possible budget-per-trial settings are taken into account with a maximum amount of resources while some fractions of the overall effort are allocated to each setting. Practically, hyperband operates by defining several "brackets", where each bracket is a SHA with a designated configuration (different initial number of configurations versus budget per configuration trade-off). Low-budget brackets could explore a certain number of configurations superficially whereas high-budget brackets possess more resources to focus on the examination of a few configurations deeply. By mixing both low-budget and high-budget brackets, Hyperband could become more robust to opt for the optimal budget requirements, thus Hyperband as a whole is comparable to the ideal SHA within a factor logarithmic in the ratio to be considered between the maximum and minimum budgets. Hyperband has been shown with empirical evidence to substantially speed up hyperparameter search for deep networks, when compared to random search. The process of boosting the performance of hyperparameter search could be done by early termination of substandard models and adoption of only a fraction of computation to find models with satisfactory performance which is comparable to random search. Obviously, hyperband is data-driven and adaptive, requiring no prior guess about how long to train each model, which is a major advantage in practice.

Plain Hyperband and SHA are limited in their applications as both treat the selection of configurations to continue as a blunt process based on only the current performance, without using any predictive modeling of the hyperparameter space. To rectify the problem, hybrid methods such as the Bayesian Optimization + HyperBand (BOHB) is employed as a proposed solution [29]. BOHB augments the Hyperband procedure with a Bayesian optimization perspective as it employs a Tree Parzen Estimator (TPE) model to guide the sampling of new hyperparameter configurations, instead of choosing them uniformly at random. The multi-fidelity racing of Hyperband is used by BOHB to evaluate a certain number of configurations, but preferentially sample configurations that past observations indicate are promising, according to the TPE model that is updated with the concomitant results. This hybrid gains the advantages of both worlds in one go: the agility of Hyperband to quickly discard bad options, and the guiding hand of Bayesian model-based search to bias the exploration towards better regions of the space. To compare tasks like tuning convolutional neural networks and reinforcement learning hyperparameters, BOHB has demonstrated more robust performance than either Hyperband or Bayesian optimization alone, especially under tight budget when pure random exploration or pure model-based exploitation can falter.

Another modern approach in this vein is the use of evolutionary algorithms with aggressive early stopping. For example, Differential Evolution HyperBand (DEHB) combines an evolutionary search strategy, i.e., differential evolution, with the multi-fidelity evaluation of Hyperband. A DE evolution is executed over hyperparameter populations while subjecting individuals to successive halving-style culling based on partial results [30]. Adopting a similar approach, one can integrate Asynchronous Successive Halving (ASHA) as a scheduler with various search algorithms; ASHA is a version of successive halving that functions without synchronizing after completing each rung and it permits trials to commence and end asynchronously, hence creating ideal distributed settings with numerous parallel workers [5]. ASHA achieves practically the same result as Hyperband but with greater efficiency in real systems because workers are never left idle to wait for others to finish a rung. These hybrid and multi-fidelity methods reflect a trend in HPO towards adaptive resource allocation rather than treating each hyperparameter trial as an all-or-nothing proposition. They possess the selective momentum to dynamically allocate more resources to promising trials and curtail the resources spent on underperforming ones.

In terms of theoretical background, multi-fidelity methods like SHA/Hyperband could work in relation to the concept of exploration versus exploitation in multi-armed bandits [28, 31]. Each hyperparameter configuration in these methods is like an arm; pulling an arm with a certain budget yields a noisy reward (partial training result). Algorithms like Hyperband implement a principled strategy to allocate pulls (training epochs) to each arm in proportions that balance discovering the best arm versus concentrating on arms that have seemingly good appearance. Convergence guarantees for these methods assume some forms of correlation between low-budget and high-budget performance, an implicit assumption that a substandard configuration after a small budget is unlikely to turn outstanding with more budget which usually holds in practice, despite the possibility of exceptions. Empirical studies on benchmarks for image classification, etc. show that multi-fidelity optimizers could reach near-optimal hyperparameter settings orders-of-magnitude faster in wall-clock time than conventional methods, particularly when training time is the bottleneck [32]. These methods have soon been adopted in both academic research and production AutoML systems because they provide straightforward solution to address the heavy computational cost of deep learning HPO by pruning uneconomical trials early.

1.1.6 Differentiable hyperparameter optimization

The emerging research in hyperparameter optimization seeks to turn the hyperparameter tuning process differentiable, thereby enabling the use of gradient-based optimization methods to adjust hyperparameters [33]. Traditional HPO treats the training of a model with a given hyperparameter set as an atomic operation that yields a final validation loss or accuracy, with no notion of gradient in relation to hyperparameters, since the hyperparameters are not part of

the end-to-end computation graph. Differentiable hyperparameter optimization methods formulate HPO as a bilevel optimization problem, in which the weights of the model are the inner-level variables and the hyperparameters are the upper-level variables to be optimized by differentiating during the training process [34].

One approach to differentiable HPO is to approximate the training procedure in a differentiable manner. For instance, one can train the model for a few steps and then use implicit differentiation or finite differences of the resulting validation loss to estimate gradients with respect to hyperparameters like learning rate or regularization strength [35, 36]. According to some pioneering works, certain hyperparameters including weight decay, or a learning rate schedule parameterized by a few variables could compute the gradient of the validation loss with respect to those hyperparameters. During the process, a truncated training loop is unrolled or closed-form solutions in simpler models could be used and then the hyperparameters are updated via gradient descent [37, 38]. Another approach involves relaxing discrete hyperparameters into continuous variables using a softmax or continuous relaxation technique. For example, methods inspired by differentiable neural architecture search, such as DARTS, create a weighted mixture of hyperparameter settings like a soft selection between alternative layer configurations or data augmentations. These methods apply gradient descent to adjust those weights, effectively identifying the optimal choices of hyperparameter [39]. Once training is complete, the continuous relaxation is projected back to a discrete choice by picking the argmax weight as the hyperparameter choice [39, 40].

A recent innovation in this domain is to perform HPO within a single training run [35]. Instead of the traditional loop of "set hyperparameters, fully trained model, observed results", differentiable HPO can adjust hyperparameters continuously in one training cycle. As a concrete example, an approach dubbed Differentiable Hyperparameter Optimization (DHPO) has been proposed where hyperparameters of each layer (like kernel sizes, and numbers of filters, etc.) are encoded as learnable parameters alongside the weights of networks, and an outer loss on a validation set guides their updates [35, 37, 38]. Such methods have reported achieving similar or better accuracy than exhaustive HPO at a fraction of the cost. They tune several hyperparameters in one run that would normally require training dozens of separate models via random or Bayesian search, hence yielding an order-of-magnitude speedup.

While promising, differentiable HPO comes with practical challenges, the bilevel optimization is unstable or memory-intensive as it keeps track of changes in hyperparameters that would have affected the outcomes of training. Techniques like truncating the unrolled training (to a smaller number of steps) or using approximations (like using older model states to approximate the hyperparameter gradient) are considered necessary. There is also a risk of hyperparameters overfitting the validation set if not regularized, because they are directly optimized to minimize validation error within the same run when that validation data is found. Theoretical analysis of these methods typically uses bilevel optimization theory; under certain convexity assumptions, the theory could guarantee convergence to stationary points of the validation loss with respect to hyperparameters. In non-convex settings as in the case of deep networks, one could not ensure global optimality and the methods might converge to a local optimum of the hyperparameters, though still performing well in practice.

In summary, differentiable hyperparameter optimization represents a cutting-edge trend aiming to further automate and speed up model tuning by merging the boundary between training and tuning. It complements the other methods under discussion so far; Bayesian and evolutionary methods operate at the level of discrete trials and treat the model as a black box; differentiable HPO opens the black box to peek at the influence of hyperparameters on training dynamics. This area is still undergoing active development, yet not commonly used in large-scale practices, it offers a glimpse of future AutoML systems where models could self-tune many of their own hyperparameters through gradient-driven adaptation.

1.1.7 Early stopping based on validation performance

One of the most common regularization and time-saving strategies in deep learning is early stopping. If further training is observed to degrade validation performance or yield no meaningful improvement, the training of a model will be suspended before the nominal number of training iterations (epochs) is completed [41, 42]. Typical early stopping implementation sets aside a validation dataset and monitors a relevant metric (e.g., validation loss or accuracy) after running each epoch or every few epochs. If the validation metric has not improved for a certain number of consecutive checks (say a patience parameter), training is stopped based on the assumption that the model overfits or it is noted that even when the training continues, there will not be additional benefits beyond random fluctuations. The model parameters from the epoch with the best validation score are then taken to be the final choice. This straightforward procedure effectively guards against overfitting as it inherently balances insufficient training (underfitting) and excessive training (overfitting on the training data) by using the validation set to guide the finding of an approximate sweet spot for stopping.

1.1.8 Successive halving and hyperband for early stopping

While the conventional early stopping is applied to a single training run so as to prevent the occurrence of overfitting, similar principles could be employed in hyperparameter search at a higher level when running numerous trainings. Previous sections in this study have covered the notion that algorithms like Successive Halving and

Hyperband implement systematic early stopping of multiple trials. In these contexts, early stopping is not primarily about avoiding overfitting but about efficient allocation of computational resources among competing hyperparameter candidates. The design of Successive Halving (SHA) helps save running a large fraction of trials after a few epochs and permits a selective subset to continue. This method is useful in performing automated early stopping for trials that appear least promising. Any single model might have stopped in advance before it overfits and its performance is worse than others.

The efficacy of SHA/Hyperband as early stopping strategies relies on the assumption of correlation in which a hyperparameter configuration that yields poor validation accuracy after, say 1 epoch, is likely to be subpar after 50 epochs [25]. This assumption could be upheld most of the time as some configurations might learn slowly and eventually outperform others if there is sufficient time allocation. The bracket strategy of Hyperband functions partially as a hedge against such cases by allotting certain configurations more initial budget in some brackets. If a configuration is truly a "late bloomer", one of the brackets with a larger initial budget might catch its potential whereas the most aggressive bracket might stop it too early. Thus, Hyperband is robust since it depends not just on a single stopping schedule.

These multi-trial early stopping methods directly impact the practical HPO. When tuning a deep neural network on an image classification task instead of training each candidate for a fixed 100 epochs, one could use Hyperband to train, namely 50 candidates for 10 epochs, then continue only the top 10 of those to 30 epochs, and afterwards, continue the top 3 to the full 100 epochs. The final result, i.e., the best of those 3 fully trained models, will be nearly as good as the best out of training all 50 for 100 epochs, but the total computation used is only a fraction of training 50×100 epochs (roughly, it trained 50 for 10 + 10 for 20 more + 3 for 70 more, which is about 50×10 + 10×20 + 3×70 = 500 + 200 + 210 = 910 epoch-equivalents, versus 5000 epoch-equivalents if all were trained fully). In this hypothetical theory, Hyperband used $\sim$18% of the computation of exhaustive search, hence an enormous saving. In exchange, one accepts a small risk: it might miss a configuration that would have eventually been the best because it looked mediocre in the first 10 epochs and got terminated. Empirical evidence and benchmarks suggest that this trade-off is worthwhile in most of the cases. The opportunity of missing the global optimum is low if the performance after a small fraction of training is at least somewhat predictive of final performance, as is the case in many tasks.

In summary, SHA and Hyperband generalize the idea of early stopping from "stop training when there is no improvement" (single run to avoid overfitting) to "stop training this configuration if it is unlikely to be the best" (multiple runs to allocate effort properly). Both uses of early stopping are complementary: one can and should use validation-based stopping within each training run to avoid overfitting and use SHA/Hyperband logic across runs to budget evaluation time. Modern HPO frameworks do both. For instance, they might run many trials with a scheduler like ASHA, and within each trial, the training code uses an early stopping callback for its own training convergence; whichever criterion triggers first will stop that trial (either outperformed by peers or plateaued on validation). This layering of early stopping ensures maximal efficiency in searching hyperparameters for deep learning.

1.1.9 Pruning models during training (structured and unstructured)

Pruning in deep learning refers to techniques that remove parameters or structures from a neural network that are deemed unnecessary for making predictions, thus yielding a smaller (and often faster or more efficient) model with minimal loss in accuracy [43, 44]. Pruning is generally implemented subsequent to or during the training process to achieve model compression or to diminish computational requirements.

Two broad categories of pruning are in practice and they involve unstructured pruning and structured pruning. Unstructured pruning removes individual weights from the weight matrices of the network (i.e., setting certain weight values to zero and eliminating their influence effectively). This leads to a sparse network: the original architecture/number of layers and neurons remains constant but many connections have zero weight and can be omitted from computation [45]. A common criterion for unstructured pruning is magnitude; in other words, after training a model, one could remove all weights whose absolute values are below a certain threshold or simply remove the smallest X% of weights globally or per layer, on the intuition that those contribute least to the output [46].

Structured pruning removes higher-level structures; for example, entire neurons in a fully-connected layer, entire channels or filters in a convolutional layer, or whole attention heads in a transformer model [47–49]. The architecture of the network in structured pruning is changed effectively; therefore, if we remove 2 out of 64 filters in a convolutional layer, the input channels of subsequent layers are reduced. Structured pruning is advantageous in yielding a smaller dense network, which could directly translate to faster inference and less memory usage without the support of special hardware. The criteria for structured pruning include metrics like the l_1 or l_2 norm of filter weights (filters removed with the smallest norm, assuming they have least contribution), or more sophisticated importance such as the amount of the loss increased when a designated neuron is ablated [50, 51]. Structured pruning is generally used with caution because removing a whole unit could exert more significant impact on network function rather than removing a single weight. It is therefore common to retrain or fine-tune the network after pruning to enable the remaining units to compensate for the lost ones. When done iteratively to prune a little and retrain, even structured

pruning could remove a significant fraction of units (e.g., 30-50% of filters) with only minor accuracy dropped in many vision models [48, 52].

When pruning is applied during training rather than as a post-processing step, it involves a schedule to start training the model with full capacity. Connections are pruned away at certain points or gradually. One method might be adopted to begin pruning after a few epochs; once the network has acquired some useful weights, every epoch removes the lowest-magnitude at 2% of the remaining weights until a target sparsity level is achieved by the end of the training. This approach, sometimes called gradual pruning, gives the network time to adjust to the decreasing capacity [46, 53]. It has been observed that if pruning is too aggressive too early, the training could deteriorate because the model might lose too many degrees of freedom before solidifying the essential patterns. Gradual pruning could integrate naturally with the training loop and could save training time in later epochs since the model becomes sparser and cheaper to evaluate, although initially one pays the cost of training the full model for some duration. Some recent strategies even do pruning from initialization or very early training: in the Lottery Ticket Hypothesis, there exists a subnetwork ("winning ticket") within a randomly initialized dense network, so full accuracy could be reached if trained in isolation [54]. In this light, algorithms have been proposed to prune networks at initialization or after just a few epochs, to identify these winning subnetworks and then train only them to go forward, and to effectively train a much smaller network as soon as possible. This yields huge training speedups because the majority of weights are never even updated [55, 56]. However, it is challenging to reliably find such subnetworks without testing many combinations, and this remains an area for further research [57, 58].

In the context of hyperparameter optimization, model pruning could be viewed as another dimension of optimization. One might consider the sparsity level or prune percentage as a hyperparameter to tune (trading off model size versus accuracy) [59]. HPO methods have been applied to pruning to find the optimal layer-wise pruning ratios that yield the best accuracy-speed tradeoff. Furthermore, incorporating pruning into HPO could accelerate the evaluation of hyperparameters; therefore, if one hyperparameter setting consistently leads to a large amount of redundant weights, pruning during training could reduce computation and thus indirectly speed up the HPO process. Some HPO frameworks include built-in "pruners"; in that context, pruner means an algorithm which could stop underperforming trials as in Hyperband rather than pruning network weights. Yet the philosophy is analogous as it cuts off the unneeded computations [27]. Both forms of pruning aim to focus resources on the most important aspect.

1.1.10 Adaptive resource allocation for tuning

Adaptive resource allocation refers to strategies that dynamically determine the distribution of computational budget across various training runs or parts of training, based on ongoing results. Several instances of this concept have been encountered since the logic of Hyperband to allocate more epochs to promising trials, or the focus of PBT on promising members of a population are prime examples. Researchers have, beside those instances, explored various techniques to fine-tune resource allocation in hyperparameter searches and training processes.

One effective yet simple heuristic is the median stopping rule attempted in some industry AutoML platforms [60, 61]. In each trial of this approach, there are certain checkpoints like after each epoch or after a fixed wall-clock time; one could compute the median of the performance metric (e.g., validation accuracy) across all completed trials at the checkpoints. If the trial in question is performing worse than this median, it is stopped early. The intuition is that if a configuration is lagging behind the typical progress of other configurations, it is unlikely to become the best by the end, so resources are better spent elsewhere. This rule is adaptive and non-parametric, adjusting to the distribution of performance in the current experiment. It is less aggressive than Hyperband as this method stops roughly the bottom half rather than a predetermined fraction, but could be combined with other methods.

Another sophisticated approach is learning curve extrapolation, which is to train all trials for a minimal amount, e.g., a few epochs, and fit a model to the trajectory of performance of each trial over time. Techniques like Bayesian modeling of learning curves or parametric models (e.g., assuming performance follows a power-law or exponential saturation curve) allow one to predict the final performance of a model, given its partial learning curve [62]. Using these predictions, decision could be made promptly to stop those trials that are predicted to have a low final accuracy, even at the current epoch, they might not be the worst. Such approaches could sometimes identify "slow starters" that will eventually perform better to avoid the pitfall of naive early stopping. However, predicting learning curves reliably is challenging owing to the variety of curve shapes neural nets, i.e., some might plateau; others might jump after a certain epoch due to learning rate schedules. Current research has shown that given apt calibrated models and sufficient prior data, learning curve prediction could further boost the efficiency of HPO on top of methods like Hyperband via catering the stopping criterion to the behavior of each specific trial rather than using a one-size-fits-all schedule.

Adaptive resource allocation extends to methods like fidelity scheduling, where one might dynamically decide not just the duration of training, but aspects like resolution of input data or model size to be used for intermediate evaluations [26]. An HPO process might first try configurations on a smaller version of the dataset or a smaller network, and progressively increase the difficulty with full data or larger model for configurations that perform equally

well in the smaller setting. This is akin to multi-fidelity, where fidelity could mean dataset size or model complexity but not just training epochs. The allocation of resources in this manner might quickly eliminate hyperparameters that just excel in small-scale tests and focus on those that could also function equally satisfactorily on larger scales.

Some recent work, on the algorithmic front, has applied reinforcement learning to the problem of resource allocation in HPO. A reinforcement learning agent observes the state of ongoing trials including their performances and remaining budgets to derive a policy to select the trial to extend or to allocate a new batch of resources, with the reward of attaining a high final model accuracy under a fixed total resource expense. This is a complex scenario but conceptually could outperform static policies by effectively allocating resources based on previous experience gained from a series of problems encountered. The agent might learn that certain patterns in early validation improvements are strong indicators of eventual success or failure, and so allocate budget accordingly.

1.1.11 Bounding Box Tuner (BBT) in context

The Bounding Box Tuner (BBT) is an innovation in hyperparameter optimization that builds upon the above approaches by introducing a dynamic space refinement mechanism [63, 64]. At its core, BBT adapts the search space itself during the optimization process, effectively tightening the bounds, i.e., the "bounding box" in hyperparameter space around regions with promising hyperparameters. This approach contrasts with most traditional HPO methods, which keep the hyperparameter search domain static throughout the search. As evidence accumulates, the search space shrinks, BBT could successfully focus computational effort on the most relevant areas and exclude regions that are unlikely to contain good solutions [65, 66].

The proposed Bounding Box Tuner might practically operate as follows: it commences with user-specified broad ranges for each hyperparameter (e.g., learning rate in [1e-5, 1e-1], number of layers in [2, 10], etc.). It then samples and evaluates a number of configurations by using random sampling or another strategy initially. Based on the results, BBT identifies a subset of the space where high-performing configurations are concentrated. The most optimal outcomes achieved to date might be achieved when learning rate is between 1e-3 and 5e-3 and the number of layers is between 4 and 6. It then updates the search bounds to that sub-region, possibly with some margins to allow exploration around it. Subsequent hyperparameter proposals are restricted to this smaller hyper-rectangle region. As the search progresses and more data points are gathered, the bounding box could be further adjusted, either contracted or slightly shifted, to continuously home in on the area yielding the best results. This iterative bounding process continues until the algorithm converges or a maximum number of trials is reached and the outcome is a progressively narrowing search that zooms into the likely optimum.

BBT complements existing HPO methods rather than replacing them as it is considered an outer loop around any base optimizer. Bayesian optimization or evolutionary search as the internal method could suggest new points after each iteration or a set of iterations and apply the bounding heuristic to adjust the domain fed into that internal optimizer [64]. Through this approach, BBT provides a form of memory or meta-learning that standard optimizers lack as it obviously remembers the region of interests. Bayesian optimization implicitly performs similar function as its surrogate by becoming very flat/uncertain in unvisited regions and sharper around visited good regions; however, it will still occasionally sample the far corners of the original space if the acquisition function visualizes the potential. BBT would outright eliminate those corners from consideration after the optimum is ascertained to lie elsewhere, thereby potentially saving the exploratory evaluations. BBT is exploitative to a certain extent since this technique assumes that once a region with good hyperparameters is found, the global optimum will be nearby, yet this conjecture is to be confirmed. The implementation of BBT might allow some mechanisms to occasionally test beyond the current bounds so as to avoid local optimum traps, akin to a cooling schedule in simulated annealing, so the search is sometimes allowed to escape the current region.

The idea of narrowing search bounds has antecedents in iterative deepening and zooming algorithms in global optimization. The DIRECT algorithm (DIviding RECTangles) systematically partitions the search space into increasingly smaller hyper-rectangles focusing on regions with low function values. Bounding Box Tuner brings a similar spirit to hyperparameter tuning but in a more heuristic-driven way, is tailored to machine learning. The operation works under the assumption that users begin with rather wide hyperparameter ranges, but not all of that space is indeed reasonable. By identifying and zooming to an identified region, BBT effectively performs a form of automated range tuning, which is manually prepared by practitioners. Subsequent to a preliminary search, one might realize that "all the best models had learning rate $\sim$0.001, so let's search more closely around that value". BBT formalizes and automates this intuition to reveal more intriguing findings.

In the context of pruning and early stopping, the philosophy of BBT is analogous. Since unnecessary model parameters are pruned or unpromising trials are stopped to save resources, BBT prunes away swathes of hyperparameter space that appear unpromising. BBT might use early-stopped results from many configurations to identify the focused region. It could run a Hyperband-like procedure on the full space to get an initial mapping of performance, then define a bounding box around the top performers and continue the search with Bayesian optimization in that reduced space for more fine-grained exploration. Such a two-phase approach, global exploration

preceding local exploitation, is not novel, but BBT provides a systematic way to carry out the transition by modifying bounds.

The introduction of BBT extends the hyperparameter optimization toolkit by emphasizing adaptive search space shaping and is particularly useful in high-dimensional problems or cases with mixed continuous and categorical parameters, where certain combinations of categories and ranges of values are recognized to be irrelevant or suboptimal. By cutting those out, BBT could reduce the dimensionality or effective volume of the search, turning subsequent optimization simpler and potentially improving the convergence speed. BBT is viewed as imposing a kind of prior or trust region that becomes tighter over time, which may assist surrogate-model-based optimizers like GPs in avoiding the distraction by too-large domains where their models are uninformative.

The drawback of BBT is its inherent incompetence to guarantee the finding of the global optimum; if the initial bounding decision is too aggressive, it might exclude the authentic best early on. Therefore, an important aspect of this innovation is the way for it to decide the timing and degree of bounding. Statistical tests or heuristic thresholds could be used to only shrink bounds when there is high confidence that the best lies within a certain interval for each hyperparameter. For example, if all top N performers have a learning rate in a narrow range, with none coming close outside that range. BBT should therefore be configured to retain some exploration, like allowing an occasional point outside the box, until late in the search.

To sum up, BBT fits into the landscape of HPO and training optimization as a higher-level strategy that can work alongside Bayesian, random, evolutionary, and multi-fidelity methods. It complements these methods by addressing a different facet, i.e., the adaptation of the search space. The attempt to overcome one of the practical challenges in HPO requires the initial guess of a reasonable search range, and it can potentially accelerate the finding of the optimum by discarding irrelevant regions earlier. This approach extends existing work by pushing more adaptivity into the HPO process, and early reports indicate that it can lead to quicker convergence in tuning tasks without sacrificing the solution quality. As with any new methods, further benchmarking (e.g., comparing BBT-augmented search versus standard search on image classification or NLP tuning tasks) will clarify its strengths and limitations. BBT is well-aligned with the prevailing trend of improving the efficiency and hands-free operation of hyperparameter optimization, hence reducing the need for manual intervention in trimming search spaces, much as algorithms like Hyperband in reducing the need for manual early stopping for trials.

1.2 Research Gaps

1. Few algorithms progressively tighten the search domain using only observed validation rewards, eliminating the need for fitted surrogate models while refocusing exploration.
2. Current mainstream methods either mutate dimensions independently (e.g., univariate TPE splits) or assume axis-aligned Gaussian priors, failing to exploit the empirical observation that the best-performing configurations often cluster within a *convex hull* defined by a small subset of elite trials.
3. Techniques that aggressively narrow the domain could converge prematurely, whereas methods that preserve global sampling throughout the search squander evaluations after the optimization has explicitly localized promising basins. A principled, budget-aware decay schedule for exploration probability remains underexplored.
4. Although pruning becomes the standard to save compute, a certain number of adaptive search algorithms do not explicitly incorporate rung-level statistics, including the median and percentile, into their decision logic, hence curtailing additional savings.

1.3 Motivation

Hyperparameter optimization drains time and energy from deep-learning projects because the dominant methods either progress too slowly or waste compute on the poor regions of the search space. Bayesian and TPE-style optimizers spend a large fraction of their wall-clock budget updating surrogate models whose cost increases in accordance with every additional dimension [67]. Random, evolutionary, or simple Local Searches avoid that overhead, yet they repeatedly sample configurations that break architectural constraints, overflow GPU memory, or situate far away from any promising area [68]. Clever early-stopping schedules reduce training time only, yet nothing could be achieved to obliterate the bad configurations to be proposed for saving purpose.

BBT is designed for situations where practitioners have only a handful of hours or GPUs to identify a competitive set of hyperparameters and where the model architecture imposes hard relationships among them. The algorithm depends on a straightforward but powerful idea: after each evaluation, it withholds the two best configurations, draws a tight hyper-rectangle that encloses both, and chooses the next trial from inside that box most frequently while occasionally exploring the global space. Every interior point inherits the feasibility of its anchors, divisibility or memory constraints are respected automatically, and the search immediately avoids vast and unproductive regions. The box shrinks whenever new top performers appear, so BBT continuously refines its focus without fitting any surrogate and without any handcrafted scheduling heuristics.

This localization strategy, when applied, delivers nearly all the accuracy of surrogate based approaches and finishes in significantly less wall-clock time, the algorithm matched Optuna-TPE on a simple MLP while consuming approximately two thirds of the time budget; it is discovered to outperform all baselines on a constraint heavy Tiny Vision Transformer (TinyViT) within the same trial limit. These results exhibit that BBT turns the question "how many configurations can we afford to try?" into "how quickly can we zoom into the right neighborhood?", hence offering teams a pragmatic way to obtain near-optimal hyperparameters before their compute allocation or iteration window expires.

1.4 Innovation and Contributions

1. **Top-2 Bounding-Box Contraction.** After an initial quasi-random seeding phase, BBT identifies the two highest-scoring configurations and defines an axis-aligned *bounding box* whose limits are their dimension-wise minima and maxima. All exploitative samples are drawn uniformly from this convex hull, thereby preserving cross-parameter interactions without expensive kernel learning.
2. **Probabilistic Global Exploration with Budget-Aware Decay.** A time-dependent exploration probability $p(t)$ (35% in the first half of the budget, 10% thereafter) injects uniformly random configurations from the original hyperparameter space. This schedule ensures continual diversity in advance and efficient exploitation later, delivering consistent accuracy gains with lower variance than fixed-rate or purely greedy strategies.
3. **Pruning-Aware Objective Evaluation.** BBT delegates training to a user-defined evaluation function that returns zero reward upon rung-based pruning (e.g., median-of-rung). This simple convention lets BBT inherit the cost savings of Hyperband-style early stopping without modifying the algorithm's core logic.
4. **Empirical Evidence of Efficiency.** Both a two-layer MLP and a TiNX-YiT on MNIST, BBT attains validation accuracies matching or exceeding Optuna-TPE, scikit-optimize GP, evolutionary search, and local random-neighbor search while reducing wall-clock tuning time by 15–50%. These gains crucially persist despite strict five-epoch evaluation windows, underscoring BBT's suitability for rapid-prototyping scenarios.

These contributions altogether address the pressing needs for efficient and compute-conscious HPO and lay the foundation for future extensions that could couple bounding-box contraction with curvature-aware surrogate modelling or trust-region Bayesian optimization.

2 Methods

2.1 Bounding Box Tuner (BBT)

BBT is a derivative-free optimizer that contracts its sampling region around the two best performing configurations observed to date [69, 70]. The algorithm proceeds as shown in Table 1. An initial quasi-random seed of $n_0 = 10$ trials populates a leaderboard sorted by validation accuracy. The two leaders, θ_1^* and θ_2^*, define an axis-aligned hyper-rectangle (the "bounding box") whose edges are the component-wise minima and maxima of these anchors. Subsequent proposals are drawn either uniformly inside this box (focused search) or uniformly from the global space Ω (exploration) with a probability that decays linearly from 0.35 to 0.10 over the course of the trial budget. Each candidate is evaluated under identical early pruning rules; if 30 consecutive proposals fail to improve the performance, the search halts early at each epoch. Early stopping will be applied with the median validation accuracy of the completed runs to be computed; trials falling below this median will be terminated. Trials that are pruned will receive a score of 0.

Table 1. Algorithm of Bounding Box Tuner

Input	**Search space Ω of d hyperparameters; budget $T\cdot$ trials; initial samples $n_0 = 10$; exploration rates $p_0 = 0.35 \to p_1 = 0.10$; patience $\tau = 30$.**
1- Initialize	Sample no configurations $\{\theta^1, \ldots, \theta^{n_0}\} \sim \mathrm{U}(\Omega)$; evaluate each; sort leaderboard by score s.
2- Select Anchors	Let $\theta^*{}_1, \theta^*{}_2$ be the top two configurations.
3- Build Box	For every dimension j : lower $_j = \min(\theta^*{}_{1j}, \theta^*{}_{2j})$; upper $_j = \max(\theta^*{}_{1j}, \theta^*{}_{2j})$.
4- Sample	Draw $\theta \sim \mathrm{U}(\Omega)$ with probability $p = \mathrm{linear}(p_0 \to p_1)$; else $\theta \sim \mathrm{U}(\mathrm{box})$. Resample until θ satisfies all hard constraints.
5- Evaluate	Train θ with embedded pruning; record $s(\theta)$.
6- Update	Insert (θ, s) into leaderboard. If $s(\theta)$ improves $\theta^*{}_1$ or $\theta^*{}_2$, replace anchor(s) and reset patience κ; otherwise $\kappa \leftarrow \kappa + 1$.
7- Terminate	Stop if $k \geq \tau$ or trials $= T$; else return to Step 3.
Output	Best configuration.

Step 1: Draw initial trials $\to$ Sample $n_0 = 10$ configurations uniformly from the full hyper-parameter space Ω

(quasi-random). Evaluate & record → Train each configuration for up to the prescribed epoch budget while applying pruning rules (e.g., median-of-rung). Create leaderboard → Store every pair (θ, s) where s is the post-pruning validation score. Sort the list in descending order of s.

Step 2: Select anchor configurations. Take the top two entries of the leaderboard $({\theta^*}_1, {\theta^*}_2)$, these two "champions" act as the corners of the current search region.

Step 3: For each dimension $j \in \{1, \ldots, d\}$:

$$\text{lower}_{\ j} = \min\left(\theta_{1*,j}, \theta_{2*,j},\right), \ \text{upper}_{\ j} = \max\left(\theta_{1*,j}, \theta_{2*,j},\right)$$

The Cartesian product of all [*lower* $_j$, *upper* $_j$] intervals forms an axis-aligned hyper-rectangle that encloses both anchors.

Step 4: Compute exploration probability

$$p(t) = p_0 - \frac{t - n_0}{T - n_0}\left(p_0 - p_1\right), p_0 = 0.35, p_1 = 0.10$$

where, t is the current trial index and T the total budget. *Draw a Bernoulli coin*;
- with probability $p_t \rightarrow$ pick a point uniformly from Ω (global exploration);
- otherwise → pick a point uniformly from the current bounding box (focused search).

Step 5: Generate a feasible candidate. Resample if necessary until the draw satisfies all hard constraints (e.g., embedding dim mod heads = 0). Every point inside the box inherits feasibility from the anchors, so violations are rare or impossible.

Step 6: Evaluate the candidate. Train under the same epoch limit and pruning policy. If the run is pruned, return a score $s = 0$; otherwise record its final validation accuracy.

Step 7: Update leaderboard and anchors. Insert $(\theta_{\text{new}}, s_{\text{new}})$ into the ordered leaderboard. If s_{new} exceeds ${\theta^*}_1, {\theta^*}_2$ replace the weaker anchor. Reset or increment the patience counter κ :
- if improvement → $\kappa \leftarrow 0$,
- else → $\kappa \leftarrow \kappa + 1$.

Step 8: Stopping criteria. Terminate when either condition holds:
- Patience counter reaches the threshold τ (no improvement in τ consecutive proposals);
- Trial budget $t = T$ is exhausted.

Otherwise return to Step 3 to rebuild the box around the current anchors and continue.

Step 9: Return result. Output the best-scoring configuration θ_{best} and its score s_{best}.

2.2 Core Features of BBT

- The sampling region contracts automatically around the empirical optima, concentrating effort where payoff is the highest while still allowing global search.
- As every new proposal is drawn between two feasible anchors, architectural divisibility constraints (e.g., embedding dimension divisible by the number of heads) are always satisfied [71].
- BBT inherits rung-based or median-of-rung pruning rules, discarding low-quality trials early and allocating compute to promising ones.
- A time decaying exploration probability prevents entrapment in the local optima, ensuring adequate coverage of Ω throughout the budget.

2.3 Dataset

The MNIST handwritten-digit corpus (60000 training, 10000 test images) is used in the experiments. Pixel intensities were normalized to zero mean and unit variance. Five thousand training images were randomly withheld for validation; no data augmentation was applied.

2.4 Models and Search Spaces

Two architectures were optimized:

- **Multi-layer perceptron (MLP).**
  - Search dimensions:
    - learning rate $LR \in [1 \times 10^{-4}, 1 \times 10^{-2}]$, log-uniform,
    - batch size $BS \in \{2^4, 2^5, \ldots, 2^7\}$,
    - hidden layers $L \in [1, 3]$,
    - units per layer $U \in [32, 256]$.
- **Tiny vision transformer (Tiny-ViT).**
  - Dimensions:

∗ $LR \in [1 \times 10^{-5}, 5 \times 10^{-3}]$,
∗ $BS \in \{2^4, 2^5, \ldots, 2^7\}$,
∗ depth $D \in [1, 6]$,
∗ embedding dimension $E \in [32, 256]$,
∗ attention heads $H \in [1, 8]$ with $E \pmod H = 0$.

Both models were trained with Adam and categorical cross-entropy.

2.5 Trial Budget and Early Stopping

Each configuration was trained for at most five epochs.

- MLP: Trials with validation accuracy < 0.30 after epoch 1 or < 0.60 after epoch 3 were pruned.
- TinyViT: At each epoch, the median validation accuracy of the completed runs was computed; trials falling below this median were terminated. Trials that were pruned received a score of 0.

2.6 Benchmarking

BBT was compared with five baselines under identical budgets of 50 scored trials per model: random search, Gaussian-Process Bayesian Optimization, Tree Structured Parzen Estimator, an $(\mu + \lambda)$ evolutionary algorithm (population=10, mutation probability=0.3) and random neighbor Local Search. Primary performance was the highest validation accuracy achieved within the budget. Secondary metrics were the cumulative search time and the number of completed epochs. All searches ran on a laptop equipped with an NVIDIA GeForce GTX 1650 Ti GPU and an Intel Core i7-10750H CPU. Wall-clock time included data loading, forward and backward passes, and optimizer overheads.

3 Results

Table 2 (MLP) and Table 3 (ViT) report the headline numbers produced by a single 50-trial sweep of each optimizer on the two architectures studied.

Table 2. Comparison on MLP

Optimizer	Best Val-Acc	Time (s)	Trials
Optuna (TPE) [72]	0.9798	2976	50
GP-Bayesian	0.9798	3271	50
Evolutionary	0.9770	3243	50
Random Search	0.9768	3102	50
Local Search	0.9608	898	16
Bounding Box Tuner	0.9788	1994	50

Table 3. Comparison on TinyViT

Optimizer	Best Val-Acc	Time (s)	Trials
Optuna (TPE)	0.9466	2572	50
GP-Bayesian	0.9436	2191	50
Evolutionary	0.9476	5173	50
Random Search	0.9482	3031	50
Local Search	0.9162	1329	22
Bounding Box Tuner	0.9492	2364	50

3.1 Multi-Layer Perceptron (MLP)

Accuracy landscape. All surrogate-based methods (Bayesian, TPE) and the performance-based method, BBT, converged within 0.20 percentage points of one another by the end of the 50-trial budget. The final score of BBT, 0.9788, was only 0.10 points lower than the joint leaders (TPE and Bayesian) and 0.20 points higher than uniform Random Search. Evolutionary search sat midway between Random Search and BBT.

Wall-clock efficiency. BBT reached its peak score after 1994 s– 982 s faster than TPE and 1277 s faster than GP-Bayesian–despite executing exactly the same number of scored trials. The algorithm therefore delivered 99.9% of the best observed accuracy at ∼34% lower cost than the nearest high-accuracy baseline. Local Search was the only faster method (898 s) but plateaued at 0.9608 after just 16 improving moves; the remaining 34 budgeted trials were left unused because the adaptive neighbor procedure could no longer locate an ascent direction.

3.2 Tiny Vision Transformer (TinyViT)

Absolute leader. In the larger, constraint-rich ViT space, BBT recorded the highest validation accuracy overall (0.9492). Random search and Evolutionary search fell 0.10-0.16 points behind, while TPE and GP-Bayesian lagged by 0.26 and 0.56 points, respectively.

Speed profile. Although GP-Bayesian completed the fastest (2191 s), its accuracy was the lowest of all surrogate-based approaches. BBT provided the best balance, finishing in 2364 s (second fastest) but producing the top score. Evolutionary search was the slowest.

3.3 Cross-Model Comparison

Consistency. BBT delivered top three accuracies in both search spaces and never dropped more than 0.10 points behind the leader. In contrast, each baseline optimizer had at least one setting where it lost more than 0.25 points to the best score. When performance is considered, BBT is cost-effective and yields better performance than other benchmarked optimization methods.

Trial utilization. BBT used the full allocation of 50 scored trials, leveraging its early stop patience counter only if no improvements occurred for 30 consecutive evaluations; this threshold was not reached in either run. Local search exhausted its local neighborhood before hitting the budget in both cases; therefore, a natural example of an optimizer that trades search reach for speed is provided.

3.4 Practical Ramifications

- Deploying BBT in place of TPE or GP-Bayesian would reduce tuning time for lightweight CNN backbones such as MLP by >1000 s while preserving sub-0.1-% accuracy differences.
- For transformer-style models with tighter architectural constraints, BBT is both the most accurate and the most time-efficient among methods that exceed 0.94 validation accuracy, rending it the obvious first choice when quick iteration cycles are paramount.
- Random search remains a viable baseline when the wall-clock time is not critical, but it requires $\sim$30% more time than BBT to reach nearly the same score on Tiny-ViT and still underperforms when BBT is on MLP.
- Evolutionary search might appeal in highly multi-modal or discrete spaces, yet its two-fold time overhead relative to BBT is not compensated by the advantages of accuracy in either benchmark.

Overall speaking, these results demonstrate that a simple axis-aligned bounding-box strategy could rival and outperform sophisticated surrogate-based hyperparameter optimizers, especially in face of stringent compute budgets and substantial architectural feasibility constraints.

4 Discussion

The present study introduces a lightweight, performance-based hyperparameter optimizer, Bounding Box Tuner (BBT) and demonstrates its competitive performance on two markedly different neural architectures. Detailed comparisons could be seen in Figures 1–3. Despite merely utilizing axis-aligned sampling and simple early pruning hooks, BBT achieved top-three validation accuracy under strict search budgets while reducing wall-clock time by hundreds to thousands of seconds relative to comparably accurate baselines. These findings highlight three notable properties of the algorithm; in other words, adaptive locality, constraint preservation, and favorable cost-benefit scaling.

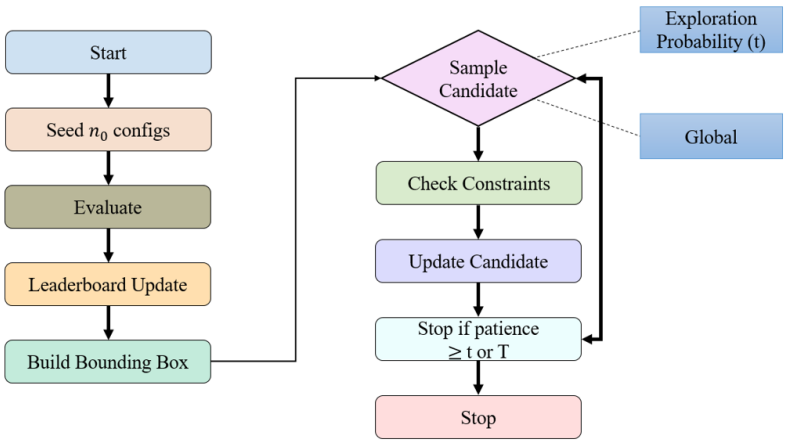

Figure 1. Algorithm of BBT

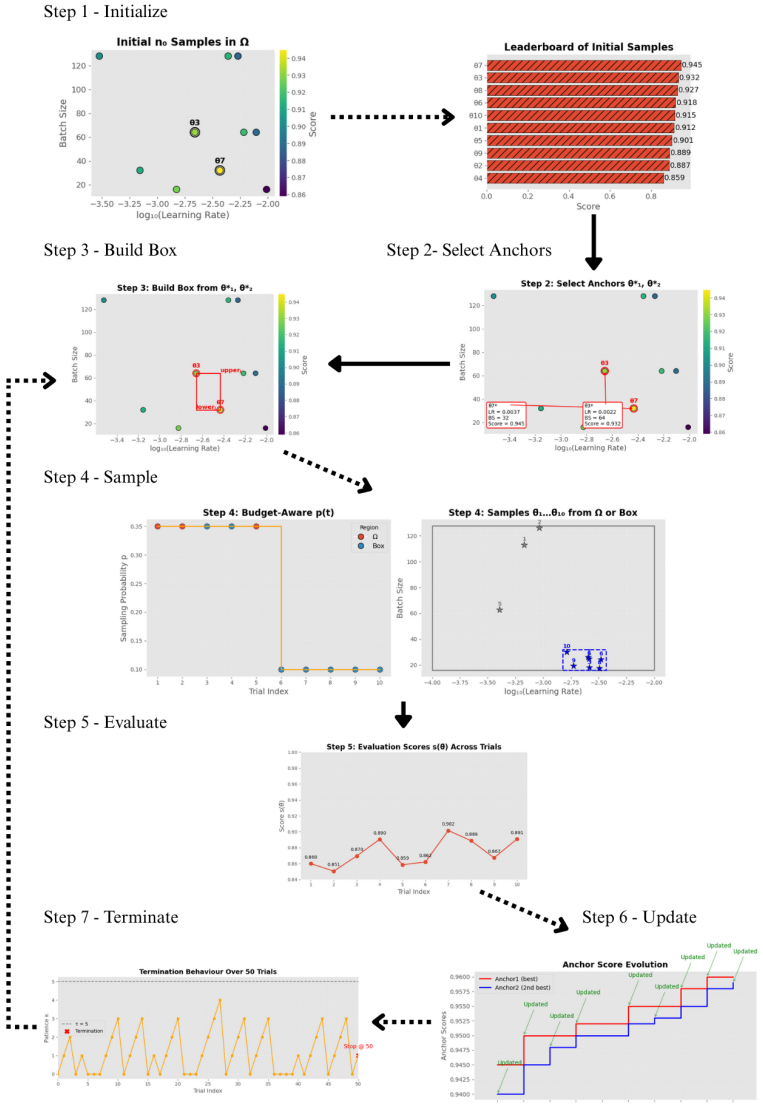

Figure 2. Algorithm of BBT

Adaptive locality without surrogate overhead. Modern Bayesian and density-estimation strategies allocate substantial runtime to fitting probabilistic models over the search space. BBT attains similar or better (As shown in Figures 3–5) end-point accuracies by exploiting a far simpler signal, i.e., the geometric span of the two best configurations. Subsequent to merely ten random seed trials, the bounding box contracts around high-payoff regions and continues to refine as new top candidates emerge. This strategy reduces both the computational overhead of surrogate maintenance and the sample complexity required to learn a useful global model, especially in low-epoch regimes where observations are noisy.

Constraint awareness. Surrogate-based optimizers such as GP-Bayesian and TPE sometimes emit transformer settings in which the embedding dimension E is not divisible by the number of attention heads H. In the implementation, these infeasible proposals are retained but assigned a validation score of 0, effectively wasting a trial. BBT avoids the issue entirely as every new candidate is sampled between two anchor configurations that already satisfy E mod $H = 0$, so feasibility is guaranteed with no additional rejection logic. This intrinsic constraint handling is valuable in hardware-bound searches that mix continuous and discrete parameters, e.g., channel counts, attention heads, or quantization widths where enforcing validity often dominates the implementation complexity.

Graceful cost scaling. BBT requires no extra meta-parameters beyond a decaying exploration rate, a patience counter, and initial sample seeds (10 in this research), and its early-stop mechanism never fires within the 50-trial budget, so larger budgets would only amortize its modest overhead further. On the MLP task, as shown in Figure 3,

BBT achieved the lowest wall-clock time per scored trial ($\approx 39\,\mathrm{s}$) among all methods, including both surrogate-based baselines. On the TinyViT task, GP-Bayesian was slightly faster per trial ($\approx 44\,\mathrm{s}$), yet delivered the lowest accuracy of the surrogate group. BBT maintained a competitive per-trial cost ($\approx 47\,\mathrm{s}$) while achieving the highest overall accuracy. Local Search remained the raw speed champion; however, in per trial timing, he is not a good option. Moreover, Local Search furnished considerably lower peak accuracy, suitably placing it for coarse prototype sweeps or extreme low-budget scenarios.

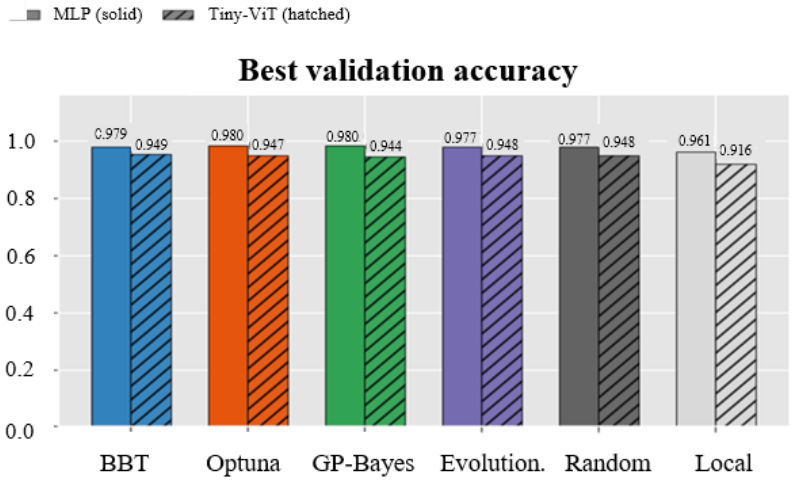

Figure 3. Validation accuracy comparison

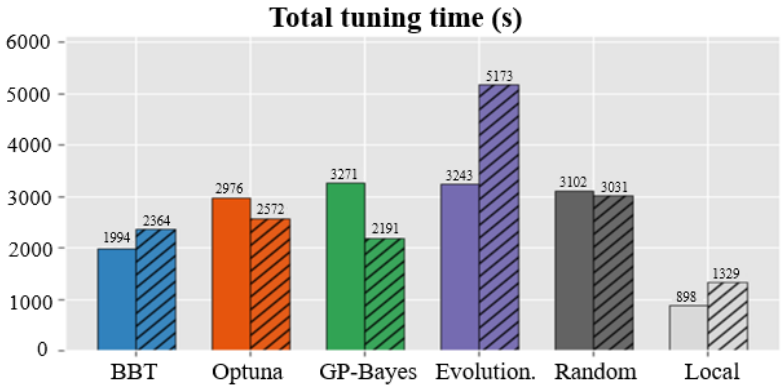

Figure 4. Total tuning time (s)

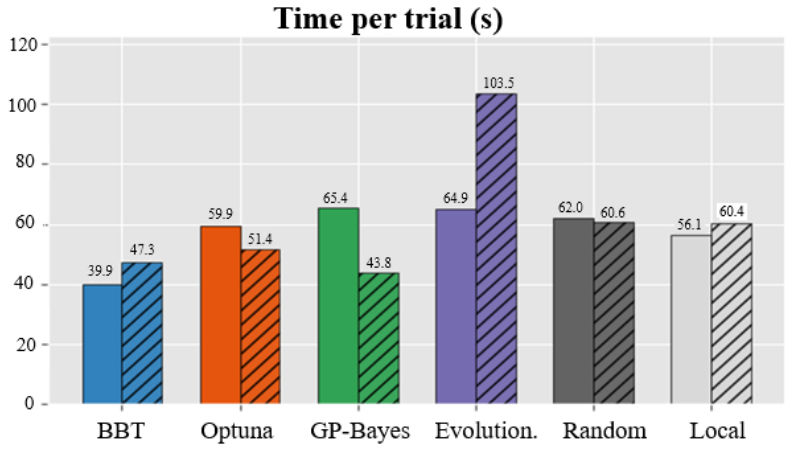

Figure 5. Time per trial (s)

4.1 Limitations

The evaluation in the current study used a single data set (MNIST) and relatively small models, limiting immediate extrapolation to large-scale vision or language workloads. In higher-dimensional spaces, the axis-aligned box might enclose substantial low-density volume, diluting the efficiency gains observed. The reliance of BBT on the top-two leaderboard positions could stall if the search landscape contains multiple disconnected basins of comparable score; periodic re-seeding or a dynamic k-best hull helps alleviate this problem. Finally, all comparisons were based on one sweep per optimizer; although the relative ordering was clear-cut, more replicates would provide tighter confidence in marginal score differences. The proposed BBT supports parallelization; however, only one GPU is available and hinders the testing of parallelization effect [73].

4.2 Practical Implications

For practitioners who have to iterate quickly under hard time budgets, BBT offers an attractive "drop-in" replacement for heavier optimizers, delivering near-state-of-the-art accuracy with minimal configuration and no surrogate tuning. Its compute savings translate directly into cost reductions on cloud platforms and shorten the feedback loop for model developers. Since the feasibility is guaranteed by construction, BBT aptly fits in the production pipelines where invalid configurations trigger expensive but failed jobs [74, 75].

4.3 Future Work

Future work includes parallelization and the effects of parallelization as there was not enough GPU to test this feature. Furthermore, the proposed BBT on more datasets and different situations could be tested, say for a simple neural network or ViT-Large, and larger hyperparameter spaces etc. This will provide a better insight of BBT against the current HPO methods. Moreover, two extensions appear particularly promising. First, a diagonal-plus-low-rank transformation of the bounding box could capture local anisotropy without sacrificing analytic simplicity, potentially boosting efficiency in the correlated subspaces. Second, integrating a lightweight performance predictor trained exclusively on in-box samples might enable hybrid exploitation-exploration schedules that retain the speed of BBT while sharpening final accuracy at the same time. Beyond the single-objective tuning, the geometric core of the algorithm naturally generalizes to multi-objective settings by defining separate bounding hulls in the objective space or by sampling along the Pareto frontier. Investigating these avenues on larger data sets and multi-node hardware will determine the full scope of the applicability of BBT.

BBT advances the hyperparameter optimization toolbox by reconciling three often competing goals: robust accuracy, strict feasibility, and low wall-clock cost. Its simplicity invites rapid adoption and further methodological refinement, thus positioning bounding box sampling as a practical bridge between naive random search and compute-intensive surrogate modelling.

5 Conclusions

Bounding Box Tuner (BBT) demonstrates that competitive hyper-parameter optimization does not require sophisticated surrogate modelling or extensive meta-parameter tuning. Based on a simple, dynamically shrinking axis-aligned hyper-rectangle and a decaying exploration schedule, BBT consistently achieved top-tier validation accuracy on both a compact CNN (MLP) and a constraint-rich TinyViT while shortening wall clock search time by 30 to 60% relative to the most accurate baselines. These gains arise from three primary mechanisms:

1. **Rapid localization.** Ten initial random probes suffice to locate a high-payoff region, after which the bounding box focuses sampling without the overhead of fitting or updating a global predictive model.
2. **Implicit constraint enforcement.** Sampling strictly within the span of two feasible anchors eliminates, by construction, invalid configurations, demonstrating a property that becomes increasingly valuable in search spaces and mixing discrete architectural choices with continuous training parameters.
3. **Compute-savvy pruning.** BBT integrates median-of-run early stopping at no extra costs, diverting resources away from unpromising trials and accelerating convergence.

Minimal parameterization of the method involves the exploration rate, patience counter and initial seeds only. It could simplify deployment in the production pipelines where ease of integration often outweighs the improvement of marginal accuracy. In addition, the favorable time accuracy profile of BBT translates directly into lower energy consumption and monetary cost on cloud platforms, an increasingly relevant factor for sustainable machine-learning practice [76].

This evaluation also highlights boundaries of the current design. Axis-aligned contraction might lose efficiency in very high-dimensional or highly anisotropic spaces; multiple near-optimal basins could trap the algorithm if the top-two solutions share a single region. Addressing these issues through adaptive box rotation, a k-best hull, or periodic re-seeding represents the next step to be followed naturally. Extending BBT to multi-objective settings and validating it on the larger-scale data sets and hardware configurations will further clarify its generalization.

Ultimately, BBT was found to be a practical middle ground between random search and heavyweight surrogate based optimizers; its simplicity to be implemented is conducive to ideal resource allocation. The running of the proposed BBT is reasonably inexpensive, intrinsically constraint-aware, and empirically competitive. All these favorable attributes recommend Bounding Box sampling to be a robust default choice for rapid and resource-constrained hyperparameter tuning in the contemporary workflows for our progressive development in deep learning.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 281–305, 2012. https://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf

[2] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," *Adv. Neural Inf. Process. Syst.*, vol. 25, 2012. https://doi.org/10.48550/arXiv.1206.2944

[3] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Advances in Neural Information Processing Systems*, vol. 24, Cadiz, Spain, 2011, pp. 1–9. https://proceedings.neurips.cc/paper_files/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf

[4] K. Jamieson and A. Talwalkar, "Non-stochastic best arm identification and hyperparameter optimization," in *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, Cadiz, Spain, 2016, pp. 240–248. https://proceedings.mlr.press/v51/jamieson16.html

[5] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, M. Ben-Tzur, M. Hardt, B. Recht, and A. Talwalkar, "A system for massively parallel hyperparameter tuning," *Proc. Mach. Learn. Syst.*, vol. 2, pp. 230–246, 2020. https://doi.org/10.48550/arXiv.1810.05934

[6] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," *Proc. AAAI Conf. Artif. Intell.*, vol. 33, no. 01, pp. 4780–4789, July 2019. https://doi.org/10.1609/aaai.v33i01.33014780

[7] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proc. IEEE*, vol. 104, no. 1, pp. 148–175, 2015. https://doi.org/10.1109/JPROC.2015.2494218

[8] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, and K. Kavukcuoglu, "Population based training of neural networks," *arXiv preprint arXiv:1711.09846*, 2017. https://doi.org/10.48550/arXiv.1711.09846

[9] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *International Conference on Learning and Intelligent Optimization*, Berlin, Germany, 2011, pp. 507–523. https://doi.org/10.1007/978-3-642-25566-3_40

[10] A. Zela, A. Klein, S. Falkner, and F. Hutter, "Towards automated deep learning: Efficient joint neural architecture and hyperparameter search," *arXiv preprint arXiv:1807.06906*, 2018. https://doi.org/10.48550/arXiv.1807.06906

[11] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowl.-Based Syst.*, vol. 212, p. 106622, 2021. https://doi.org/10.1016/j.knosys.2020.106622

[12] D. R. Jones, M. Schonlau, and W. J. Welch, "Efficient global optimization of expensive black-box functions," *J. Glob. Optim.*, vol. 13, no. 4, pp. 455–492, 1998. https://doi.org/10.1023/A:1008306431147

[13] C. K. Williams and C. E. Rasmussen, *Gaussian Processes for Machine Learning*. Cambridge, MA: MIT Press, 2006. https://www.newton.ac.uk/files/seminar/20070809140015001-150844.pdf

[14] N. Srinivas, A. Krause, S. M. Kakade, and M. Seeger, "Gaussian process optimization in the bandit setting: No regret and experimental design," *arXiv preprint arXiv:0912.3995*, 2009. https://doi.org/10.1109/TIT.2011.2182033

[15] A. D. Bull, "Convergence rates of efficient global optimization algorithms," *J. Mach. Learn. Res.*, vol. 12, no. 10, 2011. http://jmlr.org/papers/v12/bull11a.html

[16] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, Perth, Australia, 1995, pp. 1942–1948. https://doi.org/10.1109/ICNN.1995.488968

[17] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *J. Glob. Optim.*, vol. 11, no. 4, pp. 341–359, 1997. https://doi.org/10.1023/A:1008202821328

[18] R. Miikkulainen, J. Liang, E. Meyerson, A. Rawal, D. Fink, O. Francon, H. Shahrzad, A. Navruzyan, B. Duffy, and B. Hodjat, "Evolving deep neural networks," in *Artificial Intelligence in the Age of Neural Networks and Brain Computing*. Academic Press, 2024, pp. 269–287. https://doi.org/10.1016/B978-0-323-96104-2.00002-6

[19] S. R. Young, D. C. Rose, T. P. Karnowski, S. H. Lim, and R. M. Patton, "Optimizing deep learning hyper-parameters through an evolutionary algorithm," in *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments (MLHPC'15)*, Austin, TX, USA, 2015, pp. 1–5. https://doi.org/10.1145/2834892.2834896

[20] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin, "Large-scale evolution of image classifiers," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, Sydney, Australia, 2017, pp. 2902–2911. https://proceedings.mlr.press/v70/real17a.html

[21] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Upper Saddle River: Prentice Hall, 1995. https://people.engr.tamu.edu/guni/csce625/slides/AI.pdf

[22] C. Audet and J. E. Dennis Jr, "Mesh adaptive direct search algorithms for constrained optimization," *SIAM J. Optim.*, vol. 17, no. 1, pp. 188–217, 2006. https://doi.org/10.1137/040603371

[23] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural Networks: Tricks of the Trade: Second Edition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 437–478. https://doi.org/10.1007/978-3-642-35289-8_26

[24] J. Bergstra, D. Yamins, and D. Cox, "Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures," in *International conference on machine learning*, vol. 28, no. 1, Atlanta, Georgia, USA, 2013.

[25] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves," in *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, vol. 15, Buenos Aires, Argentina, 2015, pp. 3460–3468.

[26] K. Kandasamy, G. Dasarathy, J. Schneider, and B. Póczos, "Multi-fidelity bayesian optimisation with continuous approximations," in *Proceedings of the 34th International Conference on Machine Learning*, vol. 70, Sydney, Australia, 2017, pp. 1799–1808. https://proceedings.mlr.press/v70/kandasamy17a.html

[27] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Hyperband: A novel bandit-based approach to hyperparameter optimization," *J. Mach. Learn. Res.*, vol. 18, no. 185, pp. 1–52, 2018.

[28] Z. Karnin, T. Koren, and O. Somekh, "Almost optimal exploration in multi-armed bandits," in *Proceedings of the 30th International Conference on Machine Learning*, Atlanta, GA, USA, 2013, pp. 1238–1246.

[29] S. Falkner, A. Klein, and F. Hutter, "Bohb: Robust and efficient hyperparameter optimization at scale," in *Proceedings of the 35th International Conference on Machine Learning*, Stockholm, Sweden, 2018, pp. 1437–1446.

[30] N. Awad, N. Mallik, and F. Hutter, "Dehb: Evolutionary hyperband for scalable, robust and efficient hyperparameter optimization," *arXiv preprint arXiv:2105.09821*, 2021. https://doi.org/10.48550/arXiv.2105.09821

[31] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time analysis of the multiarmed bandit problem," *Mach. Learn.*, vol. 47, no. 2, pp. 235–256, 2002. https://doi.org/10.1023/A:1013689704352

[32] X. Dong and Y. Yang, "Nas-bench-201: Extending the scope of reproducible neural architecture search," *arXiv preprint arXiv:2001.00326*, 2020. https://doi.org/10.48550/arXiv.2001.00326

[33] J. Domke, "Generic methods for optimization-based modeling," in *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, La Palma, Canary Islands, Spain, 2012, pp. 318–326.

[34] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, "Bilevel programming for hyperparameter optimization and meta-learning," in *Proceedings of the 35th International Conference on Machine Learning*, Stockholm, Sweden, 2018, pp. 1568–1577.

[35] J. Lorraine, P. Vicol, and D. Duvenaud, "Optimizing millions of hyperparameters by implicit differentiation," in *Proceedings of the 23rd International Conference on Artificial Intelligence and Statistics*, Palermo, Sicily, Italy, 2020, pp. 1540–1552.

[36] F. Pedregosa, "Hyperparameter optimization with approximate gradient," in *Proceedings of The 33rd International Conference on Machine Learning*, New York, New York, USA, 2016, pp. 737–746. https://proceedings.mlr.press/v48/pedregosa16.html

[37] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *International Conference on Machine Learning*, Lille, France, 2015, pp. 2113–2122.

[38] A. G. Baydin, R. Cornish, D. M. Rubio, M. Schmidt, and F. Wood, "Online learning rate adaptation with hypergradient descent," *arXiv preprint arXiv:1703.04782*, 2017. https://doi.org/10.48550/arXiv.1703.04782

[39] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018. https://doi.org/10.48550/arXiv.1806.09055

[40] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," *arXiv preprint arXiv:1611.01144*, 2016. https://doi.org/10.48550/arXiv.1611.01144

[41] L. Prechelt, "Early stopping—but when?" in *Neural Networks: Tricks of the Trade*. Springer Berlin Heidelberg, 2002, pp. 55–69. https://doi.org/10.1007/3-540-49430-8_3

[42] R. Caruana, S. Lawrence, and C. L. Giles, "Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping," in *Advances in Neural Information Processing Systems 13*, Denver, CO, USA, 2000, pp. 402–408.

[43] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2*, Denver, CO, USA, 1989, pp. 598–605.

[44] B. Hassibi and D. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems 5*, Denver, CO, USA, 1992, pp. 164–171.

[45] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Advances in Neural Information Processing Systems 28*, Montreal, Canada, 2015, pp. 1135–1143.

[46] M. Zhu and S. Gupta, "To prune, or not to prune: Exploring the efficacy of pruning for model compression," *arXiv Preprint*, vol. arXiv:1710.01878, 2017. https://doi.org/10.48550/arXiv.1710.01878

[47] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv Preprint*, vol. arXiv:1608.08710, 2016. https://doi.org/10.48550/arXiv.1608.08710

[48] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, Venice, Italy, 2017, pp. 1389–1397. https://doi.org/10.1109/ICCV.2017.155

[49] P. Michel, O. Levy, and G. Neubig, "Are sixteen heads really better than one?" in *Advances in Neural Information Processing Systems*, vol. 32, Vancouver, Canada, 2019. https://proceedings.neurips.cc/paper_files/paper/2019/file/2c601ad9d2ff9bc8b282670cdd54f69f-Paper.pdf

[50] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," *arXiv Preprint*, vol. arXiv:1611.06440, 2016. https://doi.org/10.48550/arXiv.1611.06440

[51] J. H. Luo, J. Wu, and W. Lin, "Thinet: A filter level pruning method for deep neural network compression," in *Proceedings of the IEEE International Conference on Computer Vision*, Venice, Italy, 2017, pp. 5058–5066. https://doi.org/10.48550/arXiv.1707.06342

[52] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," in *Computer Vision – ECCV 2018: 15th European Conference*, Muich, Germany, 2018, pp. 304–320. https://doi.org/10.1007/978-3-030-01270-0_19

[53] S. Narang, E. Elsen, G. Diamos, and S. Sengupta, "Exploring sparsity in recurrent neural networks," *arXiv Preprint*, vol. arXiv:1704.05119, 2017. https://doi.org/10.48550/arXiv.1704.05119

[54] J. Frankle and M. Carbin, "The lottery ticket hypothesis: Finding sparse, trainable neural networks," *arXiv Preprint*, vol. arXiv:1803.03635, 2018. https://doi.org/10.48550/arXiv.1803.03635

[55] N. Lee, T. Ajanthan, and P. H. S. Torr, "Snip: Single-shot network pruning based on connection sensitivity," *arXiv Preprint*, vol. arXiv:1810.02340, 2018. https://doi.org/10.48550/arXiv.1810.02340

[56] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin, "Stabilizing the lottery ticket hypothesis," *arXiv Preprint*, vol. arXiv:1903.01611, 2019. https://doi.org/10.48550/arXiv.1903.01611

[57] T. Dettmers and L. Zettlemoyer, "Sparse networks from scratch: Faster training without losing performance," *arXiv Preprint*, vol. arXiv:1907.04840, 2019. https://doi.org/10.48550/arXiv.1907.04840

[58] U. Evci, T. Gale, J. Menick, P. S. Castro, and E. Elsen, "Rigging the lottery: Making all tickets winners," in *Proceedings of the 37th International Conference on Machine Learning*, vol. 119, Vienna, Austria, 2020, pp. 2943–2952. https://proceedings.mlr.press/v119/evci20a.html

[59] T. Gale, E. Elsen, and S. Hooker, "The state of sparsity in deep neural networks," *arXiv Preprint*, vol. arXiv:1902.09574, 2019. https://doi.org/10.48550/arXiv.1902.09574

[60] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google vizier: A service for black-box optimization," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Halifax, NS, Canada, 2017, pp. 1487–1495. https://doi.org/10.1145/3097983.3098043

[61] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang *et al.*, "Ray: A distributed framework for emerging ai applications," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, Carlsbad, CA, 2018, pp. 561–577. https://www.usenix.org/conference/osdi18/presentation/moritz

[62] K. Swersky, J. Snoek, and R. P. Adams, "Freeze-thaw bayesian optimization," *arXiv Preprint*, vol. arXiv:1406.3896, 2014. https://doi.org/10.48550/arXiv.1406.3896

[63] R. G. Regis and C. A. Shoemaker, "A stochastic radial basis function method for the global optimization of expensive functions," *INFORMS J. Comput.*, vol. 19, no. 4, pp. 497–509, 2007. https://doi.org/10.1287/ijoc.1060.0182

[64] D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek, "Scalable global optimization via local bayesian optimization," *Adv. Neural Inf. Process. Syst.*, vol. 32, 2019.

[65] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the lipschitz constant," *J. Optim. Theory Appl.*, vol. 79, no. 1, pp. 157–181, 1993. https://doi.org/10.1007/BF00941892

[66] B. Letham, B. Karrer, G. Ottoni, and E. Bakshy, "Constrained bayesian optimization with noisy experiments," *Bayesian Anal.*, vol. 14, no. 2, pp. 495–519, 2019. https://doi.org/10.1214/18-BA1110

[67] N. De Freitas and Z. Wang, "Bayesian optimization in high dimensions via random embeddings," *arXiv Preprint*, vol. arXiv:1309.4741, 2013. https://arxiv.org/abs/1309.4741

[68] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey," *J. Mach. Learn. Res.*, vol. 20, no. 55, pp. 1–21, 2019.

[69] T. G. Kolda, R. M. Lewis, and V. Torczon, "Optimization by direct search: New perspectives on some classical and modern methods," *SIAM Rev.*, vol. 45, no. 3, pp. 385–482, 2003. https://doi.org/10.1137/S00361445024 2889

[70] A. R. Conn, K. Scheinberg, and L. N. Vicente, *Introduction to Derivative-Free Optimization*. Society for Industrial and Applied Mathematics, 2009. https://doi.org/10.1137/1.9780898718768

[71] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Adv. Neural Inf. Process. Syst.*, vol. 30, 2017.

[72] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, Anchorage, AK, USA, 2019, pp. 2623–2631. https://doi.org/10.1145/3292500.3330701

[73] X. Bouthillier, P. Delaunay, M. Bronzi, A. Trofimov, B. Nichyporuk, J. Szeto, A. Mohamed, C. Pal, G. Varoquaux, and P. Vincent, "Accounting for variance in machine learning benchmarks," *Proc. Mach. Learn. Syst.*, vol. 3, pp. 747–769, 2021.

[74] P. Henderson, J. Hu, J. Romoff, E. Brunskill, D. Jurafsky, and J. Pineau, "Towards the systematic reporting of the energy and carbon footprints of machine learning," *J. Mach. Learn. Res.*, vol. 21, no. 248, pp. 1–43, 2020.

[75] D. Patterson, J. Gonzalez, Q. Le, C. Liang, L. M. Munguia, D. Rothchild, D. So, M. Texier, and J. Dean, "Carbon emissions and large neural network training," *arXiv Preprint*, vol. arXiv:2104.10350, 2021. https://doi.org/10.48550/arXiv.2104.10350

[76] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for modern deep learning research," *Proc. AAAI Conf. Artif. Intell.*, vol. 34, no. 9, pp. 13 693–13 696, 2020. https://doi.org/10.1609/aaai.v34i09.7123