PurGE: Towards Responsible Artificial Intelligence Through Sustainable Hyperparameter Optimization

Gauri Vaidya[1,2][a], Meghana Kshirsagar[1,2][b] and Conor Ryan[1,2][c]

[1]*Department of Computer Science and Information Systems, University of Limerick, Ireland*

[2]*Lero the Research Ireland Centre for Software, Ireland*

{*gauri.vaidya, meghana.kshirsagar, conor.ryan*}*@ul.ie*

Keywords: Grammatical Evolution, Hyperparameter Optimization, Machine Learning, Deep Learning, Search Space Pruning, Energy Efficient Computing.

Abstract: Hyperparameter optimization (HPO) plays a crucial role in enhancing the performance of machine learning and deep learning models, as the choice of hyperparameters significantly impacts their accuracy, efficiency, and generalization. Despite its importance, HPO remains a computationally intensive process, particularly for large-scale models and high-dimensional search spaces. This leads to prolonged training times and increased energy consumption, posing challenges in scalability and sustainability. Consequently, there is a pressing demand for efficient HPO methods that deliver high performance while minimizing resource consumption. This article introduces PurGE, an explainable search-space pruning algorithm that leverages Grammatical Evolution to efficiently explore hyperparameter configurations and dynamically prune suboptimal regions of the search space. By identifying and eliminating low-performing areas early in the optimization process, PurGE significantly reduces the number of required trials, thereby accelerating the hyperparameter optimization process. Comprehensive experiments conducted on five benchmark datasets demonstrate that PurGE achieves test accuracies that are competitive with or superior to state-of-the-art methods, including random search, grid search, and Bayesian optimization. Notably, PurGE delivers an average computational speed-up of 47x, reducing the number of trials by 28% to 35%, and achieving significant energy savings, equivalent to approximately 2,384 lbs of CO_2e per optimization task. This work highlights the potential of PurGE as a step toward sustainable and responsible artificial intelligence, enabling efficient resource utilization without compromising model performance or accuracy.

1 INTRODUCTION

Optimizing hyperparameters is essential to maximize the performance of Machine Learning (ML) and Deep Learning (DL) models in numerous high-impact applications, including healthcare, object detection, and image classification (Simonyan and Zisserman, 2015). Effective tuning can improve model accuracy, efficiency, and robustness, allowing ML models to better generalize across complex datasets and real-world environments. Despite this potential, determining the best hyperparameter configurations is often challenging, with manual tuning requiring considerable expertise, time, and computational resources (Diaz et al., 2017; Yu and Zhu, 2020).

The energy consumption and environmental impact of HPO are becoming increasingly significant concerns. As ML and deep DL models grow in size and complexity, their training and optimization require substantial computational resources, leading to considerable carbon emissions. For example, optimizing a natural language processing pipeline can produce approximately 78,468 lbs of CO_2e (carbon dioxide equivalent), while neural architecture search techniques can generate up to 626,155 lbs of emissions (Strubell et al., 2019). These figures underscore the urgency of developing more resource-efficient HPO methods that balance computational demands with environmental sustainability.

Traditional HPO methods aim to automate hyperparameter selection, reducing manual effort and improving model performance. For example, Random Search (RS) (Bergstra and Bengio, 2012) and Grid Search (GS) are two widely used model-free

[a] https://orcid.org/0000-0002-9699-522X
[b] https://orcid.org/0000-0002-8182-2465
[c] https://orcid.org/0000-0002-7002-5815

Vaidya, G., Kshirsagar, M. and Ryan, C.
PurGE: Towards Responsible Artificial Intelligence Through Sustainable Hyperparameter Optimization.
DOI: 10.5220/0013262100003890
Paper published under CC license (CC BY-NC-ND 4.0)
In *Proceedings of the 17th International Conference on Agents and Artificial Intelligence (ICAART 2025) - Volume 2*, pages 622-633
ISBN: 978-989-758-737-5; ISSN: 2184-433X
Proceedings Copyright © 2025 by SCITEPRESS – Science and Technology Publications, Lda.

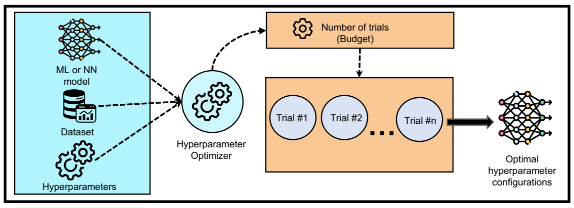


Figure 1: Traditional Hyperparameter Optimization process.

methods that blindly sample configurations from the search space. While straightforward to implement, these methods are computationally expensive and often waste resources by evaluating many suboptimal configurations. Such inefficiencies make them impractical for large, high-dimensional search spaces.

To address these limitations, more advanced model-based methods, such as Bayesian Optimization (BO) and Evolutionary Algorithms (EA), employ iterative, feedback-driven strategies to guide the search for promising configurations (Yang and Shami, 2020). These techniques balance exploration and exploitation, reducing the number of trials required to identify near-optimal hyperparameters. Although more efficient than their model-free counterparts, they still incur significant computational costs, particularly when applied to complex models with vast hyperparameter spaces. The intensity of resource in these methods highlights the need for optimization strategies that are not only effective but also computationally sustainable.

Recent advancements have focused on multifidelity optimization strategies, such as Bayesian Optimization and Hyperband (BOHB)(Falkner et al., 2018) and Differential Evolution and Hyperband (DEHB)(Awad et al., 2021). These hybrid methods integrate model-based HPO with techniques like Hyperband, which allocate computational resources more efficiently by prioritizing promising candidates and terminating evaluations of underperforming configurations early. Although these approaches improve efficiency, they still require a significant number of evaluations due to the inherent vastness of the hyperparameter search space.

One promising avenue for addressing the computational demands of HPO is search space pruning. This technique aims to reduce resource consumption by focusing computational efforts on the most promising regions of the search space, thereby minimizing evaluations of suboptimal configurations. For example, PriorBand(Mallik et al., 2023) integrates expert knowledge to prioritize high-potential regions, adaptively eliminating less promising areas. Similarly, techniques such as Successive Halving(Li et al., 2016) dynamically allocate resources to configurations with better intermediate performance, effectively pruning the search space. Other approaches, such as Learning Search Spaces for Bayesian Optimization(Perrone et al., 2019) and Hyperparameter Transfer Learning(Horváth et al., 2021), leverage historical HPO data to refine search spaces across related tasks, thus reducing computational overhead for similar models or datasets.

However, these pruning techniques often face practical limitations. Many rely on extensive prior data, which may not always be available, or make task-specific assumptions that limit their generalizability. Furthermore, heuristic-based methods or pretrained models used to predict promising regions may struggle to adapt to novel or highly complex architectures. These challenges emphasize the need for a robust, adaptive approach to search space pruning that is domain-agnostic and dynamically responsive to evolving observations during the optimization process.

This paper addresses these challenges by introducing PurGE, an innovative two-staged framework driven by Grammatical Evolution (GE). PurGE dynamically prunes the hyperparameter search space to optimize both efficiency and performance. In Stage 1, PurGE systematically narrows the search space by eliminating low-potential regions based on learned patterns. In Stage 2, it focuses on fine-tuning within the refined space to identify the optimal hyperparameter configuration. By leveraging GE, PurGE achieves a balance between exploration and exploitation, reducing computational costs without sacrificing model accuracy.

The remainder of this paper is organized as follows: Section 2 provides an overview of HPO, GE, and search space pruning techniques. Section 3 details the PurGE framework, while Section 4 outlines the experimental setup. Section 5 presents comparative results, and Section 6 discusses implications and directions for future research.

2 BACKGROUND

This section provides an overview of GE and its application in tuning the hyperparameters. It discusses recent advancements in HPO for reducing computational cost, including model pruning, dataset sampling, and search space pruning.

2.1 Grammatical Evolution

GE (Ryan et al., 1998) is a grammar-based variant of Genetic Programming that employs binary strings to represent candidate solutions. GE can evolve computer programs in any arbitrary language, provided that the language is defined using Backus-Naur Form (BNF) grammar. The process begins by mapping the genotype (binary strings) to the phenotype (computer program). The genetic operators of *crossover* and *mutation* are applied to the *population* of binary strings, with evolution progressing across successive *generations*.

A key strength of GE lies in its mapping mechanism, which offers flexibility to incorporate various grammatical structures according to specific requirements easily. GE seeks the optimal solution to a problem by maximizing or minimizing an *objective function*, with the grammar determining the set of legal structures that can evolve. Furthermore, domain knowledge can be integrated through grammar. For example, the optimization of hyperparameters in Convolutional Neural Networks (CNNs) can be expressed within the same BNF grammar used to evolve CNN architectures.

2.2 Hyperparameter Optimization

HPO problem involves selecting the optimal set of hyperparameters to maximize the performance of a model, given a learning algorithm (inducer) and dataset. Let $\mathcal{A}$ represent the learning algorithm, which induces a model M based on a set of hyperparameters h from a search space $\mathcal{H}$. Given a dataset D, we aim to find the hyperparameters h^* that maximize the performance f of the model $M = \mathcal{A}(D;h)$ induced by $\mathcal{A}$ on D:

$$h^* = \arg\max_{h\in\mathcal{H}} f(M = \mathcal{A}(D;h)), \tag{1}$$

subject to constraint functions that define the feasible region of $\mathcal{H}$:

$$a_i(h) \leq 0, \quad i = 1,2,\ldots,m,$$

$$b_j(h) = 0, \quad j = 1,2,\ldots,n.$$

In this formulation, the learning algorithm $\mathcal{A}$ acts as the *inducer* that generates the model M from D and h, with $a_i(h)$ and $b_j(h)$ representing inequality and equality constraints, respectively, to define the boundaries of the hyperparameter search space $\mathcal{H}$.

2.3 Related Works

Various approaches have been proposed to mitigate the overall computational cost of HPO (Vaidya et al., 2022; Li et al., 2016; Jamieson and Talwalkar, 2015). These strategies can broadly be categorized into three main types: model pruning, dataset sampling, and hyperparameter search space pruning.

2.3.1 Model Pruning

Neural Network Model pruning has been extensively explored since it was introduced as a solution to over-parameterized networks by Lecun et al. (LeCun et al., 1989). One widely studied technique is the Connection Sensitivity Score (SNIP) (Lee et al., 2019), which employs an initialization-based pruning method. SNIP has demonstrated the ability to prune networks effectively without significantly degrading model performance.

In addition to SNIP, Lee and Yim (Lee and Yim, 2022) proposed an alternative pruning method known as Synflow. They demonstrated that pruning can be seamlessly integrated into the HPO process, showing that the depth of neural networks does not significantly affect hyperparameter configurations. Moreover, their work highlighted that hyperparameters optimized for smaller or pruned models can be successfully transferred to larger models within the same family, such as from ResNet8 to ResNet50.

2.3.2 Dataset Sampling

Another approach to reducing the computational overhead in HPO is using subsets of datasets, rather than the full dataset, during the optimization process. DeCastro-García et al. (DeCastro-García et al., 2019) conducted a study comparing various data sampling techniques on image classification benchmarks. Their results showed that this strategy enhanced computational efficiency and maintained comparable performance to full dataset training.

Similarly, the *HyperEstimator* Vaidya et al. (2022) and *HyperGE* framework (Vaidya et al., 2023) demonstrated that fine-tuning CNNs using dataset subsets could yield results that are competitive with state-of-the-art methods, further validating the effectiveness of this approach.

2.3.3 Pruning the Hyperparameter Search Space

Reducing the hyperparameter search space has been a key focus in HPO research. *Hyperband* (Li et al., 2016), a well-known HPO framework, prunes the search space by employing early stopping of trials. In this approach, a predefined threshold is set for the number of trials, and if a trial's performance does not improve within this threshold, it is halted. Resources are then reallocated to more promising trials, allowing the system to focus its computational budget on the more fruitful configurations. This technique is particularly effective in reducing unnecessary computations and improving the overall efficiency of the search process.

Another popular technique for pruning the search space is the *successive-halving* (Jamieson and Talwalkar, 2015) algorithm. This method runs a set number of trials within a specified budget and over several iterations, evaluating their performance and discarding the worst-performing half. This process is repeated until only one trial remains.*Successive halving* efficiently narrows the search space by incrementally focusing on the most promising configurations, thus ensuring that computational resources are allocated effectively.

PriorBand (Mallik et al., 2023), a recent extension of *Hyperband*, improves upon the early stopping mechanism by using prior knowledge about the search space. This approach dynamically adjusts the budget allocation for each trial based on historical data or expert knowledge, allowing for more intelligent pruning. This enables the system to allocate resources more effectively based on prior performance, further enhancing the efficiency of the HPO process.

BOHB (Falkner et al., 2018) combines the strengths of *Bayesian optimization* and *Hyperband* to achieve more efficient search space pruning. *BOHB* leverages Bayesian optimization to model the performance of hyperparameter configurations and iteratively narrows the search space. In contrast, *Hyperband* allocates resources to the most promising configurations. This hybrid approach improves the exploration and exploitation of the search space, making it more suitable for complex models and large datasets.

Similarly, *DEHB* (Awad et al., 2021) integrates *differential evolution* with *Hyperband*, providing an efficient way to handle large-scale HPO problems. *DEHB* optimizes hyperparameters using differential evolution, while utilizing *Hyperband* for resource allocation. This combination enables more efficient search space exploration, especially for challenging optimization tasks.

Wistuba et al. (Wistuba et al., 2015) proposed another search space pruning strategy that analyzes the performance of HPO based on related datasets. By identifying non-promising areas through this analysis, the irrelevant regions of the search space can be pruned. This approach was tested on machine learning models with 19 different classifiers and showed promising results in reducing computational costs by narrowing the search to more relevant areas.

Despite the numerous advancements in reducing computational costs in HPO, the research area remains highly significant due to the complexity of machine learning models and datasets. As DL models become more sophisticated and large-scale, the search for optimal hyperparameters grows exponentially, making efficient HPO essential for practical applications. Existing methods such as pruning, early stopping, and dynamic resource allocation have shown promising results but often suffer from limitations, such as lack of dynamic adaptation to diverse model types or dataset variations. Therefore, the continued development of more adaptive and efficient search space pruning methods remains a crucial challenge in HPO.

3 PurGE

This article presents PurGE, a two-stage approach to automatically tuning hyperparameters using GE. The primary objective of PurGE is to reduce the computational burden associated with large hyperparameter search spaces by focusing on high-potential regions. An overview of the proposed framework is illustrated in Figure 2.

3.1 Stage 1: Pruning the Search Space

The primary objective of Stage 1 is to systematically narrow the hyperparameter search space, focusing computational resources on the most promising regions. This stage utilizes a grammar-guided approach, leveraging 60% of the total trial budget to identify and eliminate low-performing configurations. A *trial* is defined as a single evaluation of a hyperparameter configuration for a specific model and dataset.

The pruning process is driven by two complementary statistical techniques: the *Pearson Correlation Coefficient* (r) and *Individual Conditional Expectation* (ICE) functions. These techniques analyze the relationships between hyperparameters and model performance, enabling the identification of high-potential regions within the search space.

The Pearson Correlation Coefficient, defined in Equation 2, quantifies the linear relationship between individual hyperparameters and validation accuracy. Specifically, for each hyperparameter configuration h_i and its corresponding objective function value o_i (e.g., validation accuracy), the correlation provides insights into the influence of that hyperparameter on model performance:

$$r = \frac{\sum_{i=1}^{n}(h_i - \overline{h})(o_i - \overline{o})}{\sqrt{\sum_{i=1}^{n}(h_i - \overline{h})^2 \sum_{i=1}^{n}(o_i - \overline{o})^2}} \quad (2)$$

where n represents the total number of trials, $\overline{h}$ is the mean of the hyperparameter values, and $\overline{o}$ is the mean of the objective values. Hyperparameters with high

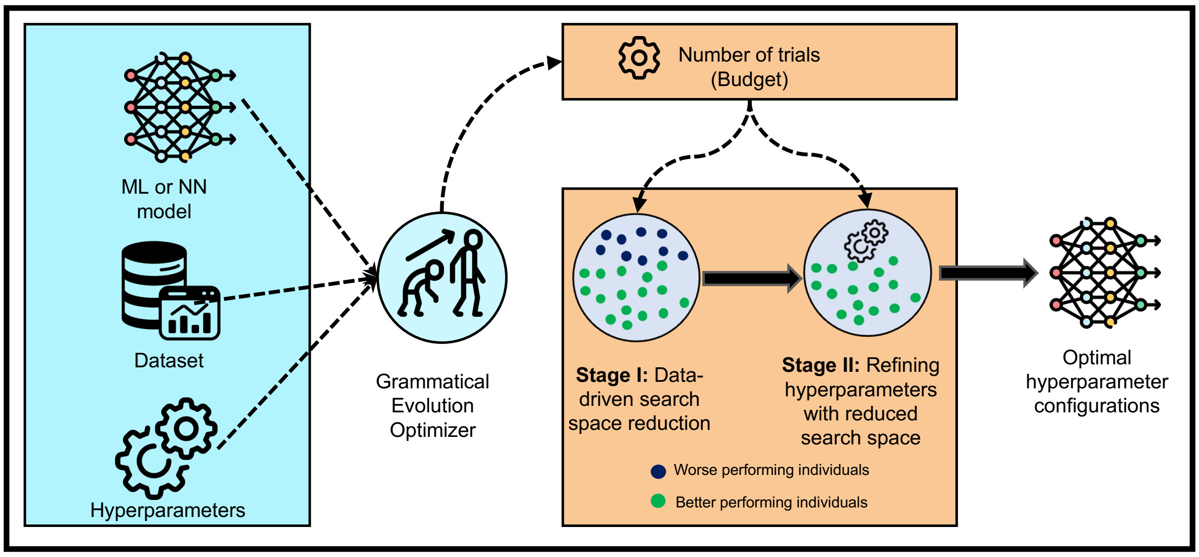

ML or NN model
Dataset
Hyperparameters
Grammatical Evolution Optimizer
Number of trials (Budget)
Stage I: Data-driven search space reduction
Stage II: Refining hyperparameters with reduced search space
Worse performing individuals
Better performing individuals
Optimal hyperparameter configurations

Figure 2: Architecture of PurGE, a two-staged Grammatical Evolution driven approach for automatically evolving hyperparameters with search space pruning.

correlation to validation accuracy are selected for further analysis.

In parallel, ICE functions are employed to extract high-performance regions for each hyperparameter. These functions identify the specific ranges within the hyperparameter space that yield superior validation accuracy, refining the focus of the search. Together, the Pearson correlation and ICE functions guide the evolutionary process by defining optimal ranges for each hyperparameter.

For instance, with a total budget of 80 trials, Stage 1 allocates 48 trials (60% of the budget) to explore the hyperparameter space, represented as a Backus-Naur Form (BNF) grammar (see Figure 6). From an initial space of 103,680 potential configurations, Stage 1 evolves 72 configurations, reducing the solution set by approximately 90%.

The pruning process operates in two phases:

1. **Correlation Analysis.** Compute the correlation between each hyperparameter and validation accuracy, producing a set $H1$, which defines preliminary bounds for promising hyperparameter ranges.
2. **Interdependence Refinement.** Identify pairs of hyperparameters with significant mutual correlation, forming set $H2$. These interdependencies further refine the bounds in $H1$, ensuring that promising configurations account for interactions between hyperparameters.

The combination of $H1$ and $H2$ results in a focused and compact search space. Figures 4 and 5 illustrate this process, while Figure 3 depicts a heatmap of hyperparameter correlations for the EfficientNet model on the CIFAR10 dataset.

3.2 Stage 2: Optimization Within the Pruned Space

Following the search space pruning in Stage 1, Stage 2 focuses on refining the search for the optimal hyperparameter configuration. By narrowing the scope to high-potential regions, computational resources are concentrated on configurations with the greatest likelihood of yielding superior performance.

This stage employs an iterative process, utilizing Grammatical Evolution to explore and evolve configurations within the pruned space. The compact search space allows for more intensive evaluation of individual configurations, enabling finer-grained optimization without incurring the computational overhead of the original space.

The algorithm dynamically balances exploration and exploitation within the reduced space, ensuring that both promising configurations and less-explored regions are considered. By iteratively evolving configurations, Stage 2 converges on the optimal hyperparameter set that maximizes validation accuracy.

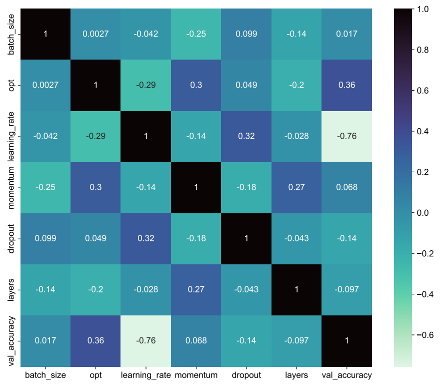
The complete algorithm, summarized below, combines the pruning strategy of Stage 1 with the focused optimization of Stage 2, offering a robust framework for hyperparameter tuning that reduces computational overhead while maintaining performance.

Algorithm 1: PurGE: Automated Hyperparameter Search Space Pruning.

```
    Input: Hyperparameter set
           $H = \{h_1, h_2, \ldots, h_n\}$, Objective
           function $f(\mathrm{VA})$, Dataset $D$
    Output: Optimal configuration $h^*$ that
            maximizes validation accuracy (VA)
 1  Step 1: Initialize Search Space: Define full
      search space $\mathcal{H}_0$ from $H$;
 2  Step 2: Stage 1 - Statistical Pruning:;
 3  Filter Low-Performing Configurations
 4      $\mathcal{H}_1 \leftarrow \{h \in \mathcal{H}_0 \mid f(h) \geq 0.5\}$;
 5  Hyperparameter-Objective Correlations
 6      For each $h_i \in H$, calculate Pearson
          correlation $r(h_i, \mathrm{VA})$;
 7      Define $H1 \leftarrow \{h_i \in H \mid |r(h_i, \mathrm{VA})| \geq \delta\}$;
 8  Inter-Hyperparameter Dependencies
 9      For each $(h_i, h_j) \in H \times H$, compute
          $r(h_i, h_j)$;
10      Define $H2 \leftarrow \{(h_i, h_j) \mid |r(h_i, h_j)| \geq \gamma\}$;
11  Extract High-Performance Ranges
12      For each $h_i \in H1$, set $R_{h_i}$ to
          top-performing values;
13      For each pair $(h_i, h_j) \in H2$, set optimal
          ranges for both $h_i$ and $h_j$;
14  Step 3: Obtain Pruned Search Space:
      $\mathcal{H}_2 \leftarrow \prod_{h_i \in H1} R_{h_i} \cup \prod_{(h_i, h_j) \in H2} R_{h_i} \times R_{h_j}$;
15  Step 4: Stage 2 - Iterative Optimization:;
16  Evolve Pruned Configurations
17      Conduct trials over $\mathcal{H}_2$;
18  Convergence Check
19      Stop when VA stabilizes or budget $B$ is
          reached;
20  Step 5: Output: Return
      $h^* = \arg\max_{h \in \mathcal{H}_2} f(h)$;
```

3.3 Example of PurGE for EfficientNet on CIFAR-10

The procedure for pruning the search space when tuning the EfficientNet model on the CIFAR-10 dataset across 48 trials is outlined. The hyperparameters under consideration include: {*batch size, optimizer, learning rate (lr), momentum, dropout, layers*}, with the objective function being validation accuracy (VA). The BNF grammar used to generate the hyperparameter configurations automatically is shown in Figure 6a. The possible combinations of each hyperparameter in Figure 6a lead to a search space of 103,680 configurations (the product of all combinations), out of which Stage 1 yields 72 configurations which are fed into Stage 2.



	batch_size	opt	learning_rate	momentum	dropout	layers	val_accuracy
batch_size	1	0.0027	-0.042	-0.25	0.099	-0.14	0.017
opt	0.0027	1	-0.29	0.3	0.049	-0.2	0.36
learning_rate	-0.042	-0.29	1	-0.14	0.32	-0.028	-0.76
momentum	-0.25	0.3	-0.14	1	-0.18	0.27	0.068
dropout	0.099	0.049	0.32	-0.18	1	-0.043	-0.14
layers	-0.14	-0.2	-0.028	0.27	-0.043	1	-0.097
val_accuracy	0.017	0.36	-0.76	0.068	-0.14	-0.097	1

Figure 3: Heatmap depicting the correlation between hyperparameters and validation accuracy for EfficientNet model on CIFAR10 dataset.

Initially, hyperparameter configurations with VA below 50% are discarded. PurGE then computes the correlation between the remaining configurations. Figure 3 illustrates the correlation heatmap between the hyperparameters and VA, revealing a negative correlation between $\{\mathrm{lr}, \mathrm{dropout}, \mathrm{layers}\}$ and VA. Based on these correlations, three hyperparameters are selected for further exploration: $\{\text{batch size}, \text{optimizer}, \text{momentum}\}$ (H1). Additionally, hyperparameter pairs with significant mutual correlation are identified, including {optimizer-momentum, lr-dropout, momentum-layers} (H2), which are also considered for further refinement.

Figure 4 is a visual representation of the pruning algorithm, PurGE. The Y-axis in the plots represents the dependence of VA on the hyperparameter configurations, with higher values indicating better performance. The shaded regions in the plots correspond to areas with high VA for each hyperparameter within H1. For example, the interpretation of the ICE plot for optimizer in Figure 4 suggests that higher partial dependence values for RMSProp indicate a potential for higher VA, while lower values for Adam and SGD correspond to lower VA.

PurGE yields the following restricted ranges for the hyperparameters:

- Optimizer: {Adam, RMSProp}
- Momentum: {0.5, 0.6, 0.7, 0.8}
- Batch Size: 128

PurGE refines the set H1 based on mutual correlation amongst hyperparameter pairs *momentum-optimizer*, *dropout-lr*, *layers-momentum*, as shown in Figure 5 to yield H2. The heatmap for the pair *layers-momentum* reveals that *momentum* values $\{0.5, 0.6, 0.7\}$ and *layer* sizes $\{96, 128\}$ exhibit high correlation. As a result, the search space for *momentum* is narrowed to $\{0.5, 0.6, 0.7\}$.

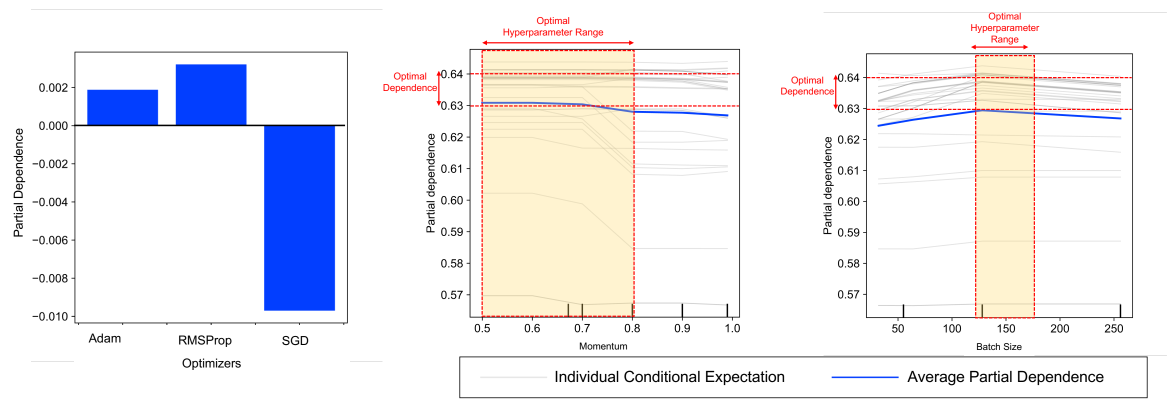


Figure 4: Explainable PurGE for search space pruning, visualized with Individual Conditional Expectation (ICE) plots. The plot illustrates the effect of hyperparameters of the EfficientNet model on the CIFAR-10 dataset, with validation accuracy on the Y-axis. Higher values on the Y-axis indicate better performance. The shaded region represents the Region of Interaction (ROI), highlighting the area where optimal performance is achieved.

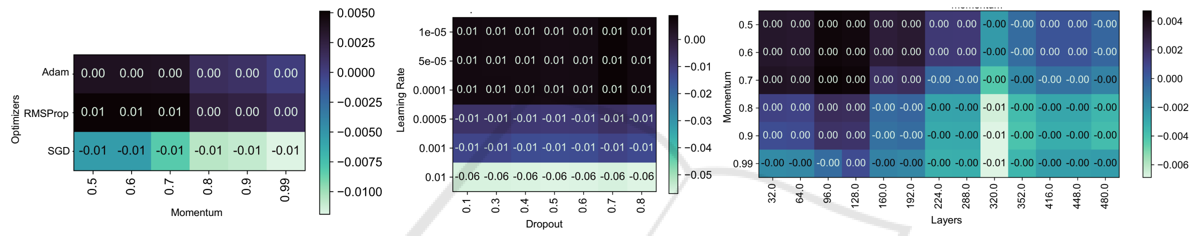


Figure 5: Heatmaps for hyperparameter pairs {`optimizer-momentum, lr-dropout, momentum-layers`} for the EfficientNet model on the CIFAR-10 dataset.

The final pruned search space for each hyperparameter is as follows:

- Batch size: 128
- Momentum: {0.5, 0.6, 0.7}
- Optimizer: RMSProp
- Learning rate: {1e-05, 5e-05, 0.0001}
- Layers: {96, 128}
- Dropout rate: {0.1, 0.3, 0.4, 0.8}

This results in only 72 unique hyperparameter combinations (the product of all possible values for each hyperparameter), representing a significant reduction of the solution set by approximately 90%.

In Stage 2, the remaining budget of 32 trials is allocated in order to determine the final hyperparameter configuration from the 72 configurations identified in Stage 1. This configuration is then used to train the model.

In this way, given a model and dataset, PurGE automatically yields optimal configurations, while its two-stage approach ensures explainability, effectively eliminating the black-box nature of the process.

4 EXPERIMENTAL SETUP

The goal of the experimental setup is to address the following research questions:

RQ1: How does pruning the search space and models impact the performance of hyperparameter optimization?

RQ2: How does pruning the search space and models affect resource utilization during hyperparameter optimization?

4.1 Datasets Details

We conducted experiments using two standard benchmarks in image classification: CIFAR10 and CIFAR100. These datasets are widely used in the deep learning community for evaluating model performance. CIFAR10 consists of 60,000 RGB images categorized into ten classes, while CIFAR100 contains the same number of images divided into 100 classes. Each dataset was partitioned into training, validation, and testing subsets in a 60:20:20 ratio.

In addition to image datasets, we included tabular datasets to evaluate PurGE on non-image tasks. These datasets were selected to have no missing values, ensuring the HPO process was not influenced by

```
<model> ::= <hyperparameters>

<hyperparameters> ::= <batch_size> <dropout_rate> <num_layers> <optimizer>
                      <learning_rate> <momentum>

<batch_size> ::= 32 | 64 | 128 | 256

<dropout_rate> ::= 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9

<num_layers> ::= 32 | 64 | 96 | 128 | 160 | 192 | 224 | 256 |
                 288 | 320 | 352 | 384 | 416 | 448 | 480 | 512

<optimizer> ::= adam | sgd | rmsprop

<learning_rate> ::= 0.00001 | 0.0001 | 0.001 | 0.01 | 0.1 |
                    0.00005 | 0.0005 | 0.005 | 0.05 | 0.5

<momentum> ::= 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 0.99
```

(a)

```
<model> ::= <hyperparameters>

<hyperparameters> ::= <learning_rate> <gamma> <max_depth>
                      <colsample_bylevel> <subsample>

<learning_rate> ::= 0.025 | 0.05 | 0.1 | 0.2 | 0.3

<gamma> ::= 0 | 0.1 | 0.2 | 0.3 | 0.4 | 1.0 | 1.5 | 2.0

<max_depth> ::= 2 | 3 | 5 | 7 | 10 | 100

<colsample_bylevel> ::= 0.25 | 0.5 | 0.75 | 1.0

<subsample> ::= 0.15 | 0.5 | 0.75 | 1.0
```

(b)

Figure 6: Search space represented as BNF grammar for (a) EfficientNet and ResNet; (b) XGBoost.

Table 2: Experimental Settings.

Parameter	Value
Runs	5
Search Algorithm	GA
Initialisation	PI grow
Selection	Tournament
Tournament Size	2
Crossover Type	Variable one point
Crossover Probability	0.95
Mutation Type	Integer flip per codon
Mutation Probability	0.01
Population Size	10
Total Generations	Stage 1: 5, Stage 2: 3

imputation techniques. The tabular datasets were also split into training, validation, and testing subsets in a 60:20:20 ratio. Table 1 summarizes the details of the datasets, including the number of instances and classes.

4.2 Models and Hyperparameters

The experiments involved three models: ResNet, EfficientNet, and XGBoost. The first two are CNNs, representing different trade-offs between performance and computational efficiency, while XGBoost is a gradient-boosting algorithm widely used in tabular data classification.

The hyperparameter search space for the models, including XGBoost and CNNs, is represented in the BNF grammar, as illustrated in Figure 6. These hyperparameters are treated as a discrete search space, which PurGE explores and compares with traditional hyperparameter optimization methods.

4.3 GE Parameters

For the PurGE framework, we utilize GE with key parameters as presented in Table 2: Genetic Algorithm (GA) for the search process, Tournament selection with a size of 2, and a variable one-point crossover with a 95% probability. Mutation occurs with a 1% probability per codon, and the population size is set to 10, spanning five generations in Stage 1 and 3 generations in Stage 2 for efficient optimization.

4.4 Baseline Methods

To benchmark the performance of PurGE, we used three popular hyperparameter optimization techniques as baselines using the `Optuna` (Akiba et al., 2019) framework:

1. **Random Search (RS).** A basic search method where configurations are randomly sampled from the defined search space.
2. **Grid Search (GS).** A more exhaustive approach that evaluates all possible combinations of hyperparameter values within a predefined grid.
3. **Tree-structured Parzen Estimator (TPE).** A BO method (Bergstra et al., 2011) that builds a probabilistic model to estimate the performance of hyperparameter configurations, guiding the search for optimal configurations more efficiently.

These baselines were used to compare PurGE's accuracy, computational efficiency, and resource utilization performance.

4.5 Training Budget

In image classification tasks, a *trial* is defined as a single hyperparameter configuration trained on the entire dataset using a pruned model as a surrogate for five

Table 1: Dataset Details

Modality	Dataset	Abbrv.	#classes	#instances	Models Employed	Abbrv.
Image	CIFAR0	C10	10	60000	EfficientNet7	EN
	CIFAR100	C100	100	60000	ResNet50	RN
Tabular	Segment	SG	7	2310	XGBoost	XB
	Waveform	WV	3	5000		
	Bank	BK	2	11163		

epochs. A *trial* corresponds to training an XGBoost model using one hyperparameter configuration over the entire dataset for tabular data classification. The fitness score for each trial is based on the model's performance on the *val* split.

The *budget* for each experiment was fixed at 80 trials as suggested in literature (Bergstra et al., 2011). After completing the 80 trials, the best hyperparameter configuration was selected and used to train the model for an additional 50 epochs. This configuration was applied to both PurGE and the baseline models.

For PurGE, the experiments were performed in two distinct stages:

- **Stage 1.** A population size of 10 with a generation count of 5, leading to 50 trials.
- **Stage 2.** A population size of 10 with a generation count of 3, resulting in 30 trials.

The number of trials in each stage is calculated as:

$$\text{Total trials} = \text{Pop size} \times \text{Gen count} \tag{3}$$

5 RESULTS AND DISCUSSIONS

This section presents the results of the experiments designed to address the research questions (RQ1 and RQ2). Specifically, the impact of search space pruning on hyperparameter optimization is evaluated in terms of performance (accuracy) for RQ1. The effect of pruning on resource utilization, such as computational time, is examined for RQ2. The results are analyzed through comparisons with baseline methods, including RS, GS, and TPE.

Table 3: P-values and Significance Interpretation (S = Significant, NS = Not Significant) against PurGE.

Model	**Dataset**	**p-values against PurGE**		
		GS	**RS**	**TPE**
XB	SG	S	S	NS
	WV	S	S	S
	BK	S	S	S
EN	C10	S	S	NS
RN	C100	S	S	S

PonyGE2 (Fenton et al., 2017), a GE implementation in Python, was adapted to run all the experiments with the Pytorch framework. All experiments were conducted simultaneously on Intel Xeon Silver 4215R CPU @ 3.20 GHz with Quadro RTX 8000 GPU.

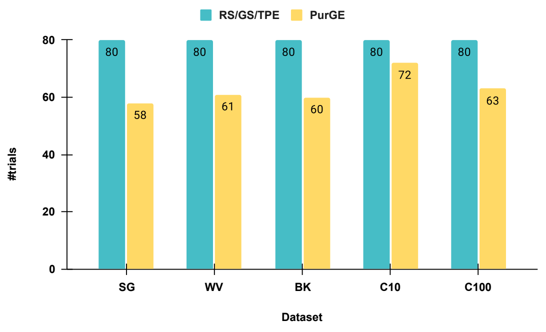


Figure 7: Benchmarking the performance of PurGE across all datasets with respect to the allocated budget, compared against baseline methods.

5.1 Impact on Performance

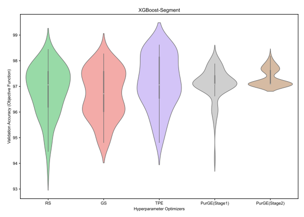
On the tabular datasets, PurGE demonstrated competitive performance compared to baseline methods. For the SG dataset, PurGE achieved an accuracy of 97.92%, while RS, GS, and TPE reported slightly higher accuracies of 98.26%, 98.44%, and 98.61%, respectively. On the WV dataset, PurGE obtained an accuracy of 87.04%, performing comparably to RS (87.6%) and GS (88.04%), while outperforming TPE, which reported an accuracy of 84.96%. For the BK dataset, PurGE achieved an accuracy of 85.34%, which was slightly lower than RS (85.77%), GS (85.45%), and TPE (85.99%).

For the image datasets, PurGE showed varying performance across tasks. On the C10 dataset, PurGE achieved an accuracy of 75.57%, which was comparable to TPE (75.57%) but slightly lower than RS (79.01%). GS, however, reported a significantly lower accuracy of 60.88%, highlighting the inefficiency of grid-based methods in this context. On the C100 dataset, PurGE outperformed the baseline methods, achieving an accuracy of 17%, while RS, GS, and TPE reported substantially lower accuracies in the range of 4-6%.

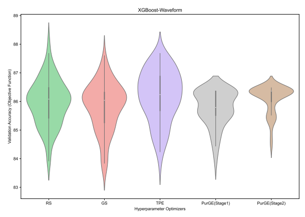
The relatively weak performance of all methods on the C100 dataset can be attributed to the limited number of samples available per class, which presents a significant challenge for hyperparameter optimization. Despite this, PurGE's ability to achieve higher accuracy on C100 underscores its potential for tackling complex, high-dimensional search spaces more effectively than traditional methods.

5.1.1 Statistical Significance

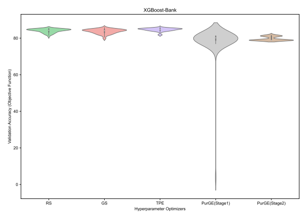
Mann-Whitney U tests with Beck and Hollern's correction were conducted to evaluate the statistical significance of the observed performance differences, as shown in Table 3. The results indicate that PurGE sig-



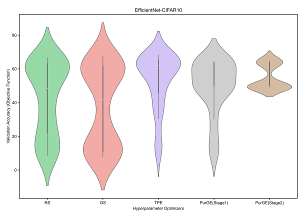
(a) XGBoost-Segment



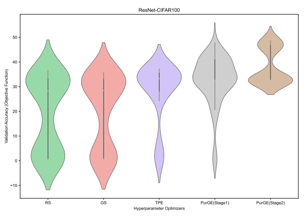
(b) XGBoost-Waveform



(c) XGBoost-Bank



(d) EfficientNet-CIFAR10



(e) ResNet-CIFAR100

Figure 8: Violin plots depicting the performance of various optimizers during the exploratory search phase.

nificantly outperforms baseline methods (GS, RS, and TPE) in many cases. Specifically, PurGE consistently achieves statistically significant improvements over GS and RS across most datasets. While the differences between PurGE and TPE vary across datasets, significant improvements are observed for SG, WV, and BK, whereas the differences for C100 are less pronounced.

Figure 8 shows violin plots of validation accuracy across multiple runs for each method. An interesting observation is the low standard deviation of validation accuracy during Stage 2 of PurGE. This indicates that PurGE focuses on a *high-yielding region* of the hyperparameter space, where multiple configurations yield near-optimal or optimal performance. The narrow distribution suggests that PurGE consistently converges on effective configurations, enhancing the reliability of the optimization process. This observation highlights the potential for further research into decision-making frameworks that can select among multiple optimal configurations based on factors such as hardware efficiency or energy consumption.

5.2 Impact on Resource Utilization

PurGE demonstrates substantial improvements in resource utilization by reducing the number of trials required for hyperparameter optimization compared to baseline methods. As shown in Figure 7, PurGE achieves a consistent reduction in the number of trials across all datasets. For example, on tabular datasets such as SG and WV, PurGE reduces the required trials by approximately 20-25%, while maintaining competitive performance. On image datasets, such as C10, PurGE achieves a similar reduction in trials, with a more significant improvement observed on C100, where baseline methods require substantially more trials to achieve lower accuracy.

The reduction in trials directly impacts computational efficiency. For instance, on datasets like SG and WV, the savings in trials translate to a reduction of computational effort by approximately 20-30%. On more complex datasets like C100, where training and evaluation are resource-intensive, PurGE completes the optimization process with fewer trials, reducing energy consumption while still achieving competitive accuracy.

Figure 9 highlights the speed-ups achieved by PurGE. On tabular datasets, PurGE achieves notable speed-ups, such as 319.91x on SG and 47.39x on WV, which are attributed to the early pruning of low-performing configurations. For image datasets, while the speed-ups are more modest (2.06x on ResNet-C100 and 2.23x on EfficientNet-C10), they are significant given the computational complexity of these tasks.

PurGE's approach of dynamically pruning the hyperparameter space allows it to concentrate resources on promising regions, reducing unnecessary evaluations. This reduction in computational overhead, combined with the consistent performance across datasets, highlights the utility of PurGE for resource-conscious hyperparameter optimization tasks.

5.3 Energy Savings with PurGE

The energy savings provided by PurGE compared to traditional methods regarding reduced carbon emis-

sions are estimated by considering the average speed-up and reduction in the number of trials. It has been reported that optimizing an NLP pipeline generates approximately 78,468 lbs of CO_2e (Strubell et al., 2019).
An average speed-up of 47x is assumed for PurGE, meaning that the optimization task can be completed in 1/47th of the time required by traditional methods, assuming energy consumption is proportional to time spent. Additionally, PurGE is reported to reduce the number of trials by approximately 28% to 35% on average. Since energy consumption per trial is assumed to be constant, this trial reduction further lowers the computational load and energy consumption. If a 47x speed-up and a 30% reduction in trials are achieved, the total Energy Reduction Factor (ERF) is approximated as:

$$\text{ERF} = 47 \times (1 - 0.30) = 47 \times 0.70 = 32.9 \quad (4)$$

Thus, the Energy Savings (ES) in terms of CO_2e emissions can be calculated as:

$$\text{Savings} = \frac{78,468\,\text{lbs of CO\_e}}{32.9} \approx 2,384.3\,\text{lbs of } CO_2e \quad (5)$$

Based on the NLP example, PurGE could save approximately 2,384 lbs of CO_2e per optimization task compared to traditional methods.

In summary, the findings show that PurGE successfully addresses both research questions by improving the efficiency and performance of hyperparameter optimization through systematic pruning of the search space. The results confirm that pruning boosts model performance and significantly reduces resource use, making PurGE a practical solution for resource-efficient hyperparameter optimization.

6 CONCLUSIONS

This article introduces PurGE, a two-stage approach for automatically tuning hyperparameters of ML and DL models through search space pruning driven by GE. PurGE achieves test accuracies that are competitive with or superior to state-of-the-art methods, including RS, GS, and BO, across all tested datasets. Notably, PurGE delivers an average computational speed-up of 47x and reduces the number of trials by 28% to 35%. Furthermore, it results in significant energy savings, equivalent to approximately 2,384 lbs of CO_2e per optimization task. These findings highlight PurGE's ability to enhance both model performance and resource utilization, positioning it as an efficient and environmentally responsible approach to hyperparameter optimization. Future work will involve benchmarking PurGE across a broader set of domains to further assess its scalability and applicability.

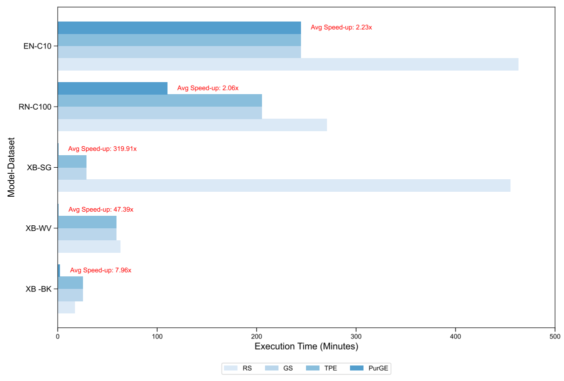


Figure 9: Speedup with PurGE against RS, BS and TPE.

ACKNOWLEDGEMENTS

This publication has emanated from research conducted with the financial support of Taighde Éireann – Research Ireland under Grant No. 18/CRT/6223.

REFERENCES

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework.

Awad, N., Mallik, N., and Hutter, F. (2021). Dehb: Evolutionary hyperband for scalable, robust and efficient hyperparameter optimization.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 2546–2554, Red Hook, NY, USA. Curran Associates Inc.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13(null):281–305.

DeCastro-García, N., Castañeda, Á. L. M., García, D. E., and Carriegos, M. V. (2019). Effect of the sampling of a dataset in the hyperparameter optimization phase over the efficiency of a machine learning algorithm. *Complex.*, 2019:6278908:1–6278908:16.

Diaz, G., Fokoue, A., Nannicini, G., and Samulowitz, H. (2017). An effective algorithm for hyperparameter optimization of neural networks.

Falkner, S., Klein, A., and Hutter, F. (2018). Bohb: Robust and efficient hyperparameter optimization at scale.

Fenton, M., McDermott, J., Fagan, D., Forstenlechner, S., Hemberg, E., and O'Neill, M. (2017). Ponyge2: grammatical evolution in python. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17. ACM.

Horváth, S., Klein, A., Richtárik, P., and Archambeau, C. (2021). Hyperparameter transfer learning with adaptive complexity.

Jamieson, K. G. and Talwalkar, A. (2015). Non-stochastic best arm identification and hyperparameter optimization. *CoRR*, abs/1502.07943.

LeCun, Y., Denker, J., and Solla, S. (1989). Optimal brain damage. In Touretzky, D., editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann.

Lee, K. and Yim, J. (2022). Hyperparameter optimization with neural network pruning.

Lee, N., Ajanthan, T., and Torr, P. (2019). SNIP: Single-shot pruning based on connecion sensitivity. In *International Conference on Learning Representations*.

Li, L., Jamieson, K. G., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2016). Efficient hyperparameter optimization and infinitely many armed bandits. *CoRR*, abs/1603.06560.

Mallik, N., Bergman, E., Hvarfner, C., Stoll, D., Janowski, M., Lindauer, M., Nardi, L., and Hutter, F. (2023). Priorband: Practical hyperparameter optimization in the age of deep learning.

Perrone, V., Shen, H., Seeger, M., Archambeau, C., and Jenatton, R. (2019). Learning search spaces for bayesian optimization: Another view of hyperparameter transfer learning.

Ryan, C., Collins, J., and Neill, M. O. (1998). Grammatical evolution: Evolving programs for an arbitrary language. In Banzhaf, W., Poli, R., Schoenauer, M., and Fogarty, T. C., editors, *Genetic Programming*, pages 83–96, Berlin, Heidelberg. Springer Berlin Heidelberg.

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*.

Strubell, E., Ganesh, A., and McCallum, A. (2019). Energy and policy considerations for deep learning in NLP.

Vaidya, G., Ilg, L., Kshirsagar, M., Naredo, E., and Ryan, C. (2022). Hyperestimator: Evolving computationally efficient cnn models with grammatical evolution. In *Proceedings of the 19th International Conference on Smart Business Technologies*. SCITEPRESS - Science and Technology Publications.

Vaidya, G., Kshirsagar, M., and Ryan, C. (2023). Grammatical evolution-driven algorithm for efficient and automatic hyperparameter optimisation of neural networks. *Algorithms*, 16(7).

Wistuba, M., Schilling, N., and Schmidt-Thieme, L. (2015). Hyperparameter search space pruning – a new component for sequential model-based hyperparameter optimization. In Appice, A., Rodrigues, P. P., Santos Costa, V., Gama, J., Jorge, A., and Soares, C., editors, *Machine Learning and Knowledge Discovery in Databases*, pages 104–119, Cham. Springer International Publishing.

Yang, L. and Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316.

Yu, T. and Zhu, H. (2020). Hyper-parameter optimization: A review of algorithms and applications.