Optimization of Learning Workflows at Large Scale on High-Performance Computing Systems

Romain Egele

université PARIS-SACLAY

Optimization of Learning Workflows at a Large Scale on High-Performance Computing Systems

Optimisation de Processus d'Apprentissage à Grande Échelle sur des Systèmes de Calcul Haute-Performance

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°580, Sciences et Technologies de l'Information et de la Communication (STIC)
Spécialité de doctorat: Informatique
Graduate School : Informatique et Sciences du Numérique. Référent : Faculté des sciences d'Orsay

Thèse préparée au **Laboratoire interdisciplinaire des sciences du numérique (Université Paris-Saclay, CNRS)**,
sous la direction de **Isabelle GUYON**, Professeur
et le co-encadrement de **Prasanna BALAPRAKASH**, Directeur de Recherche

Thèse soutenue à Paris-Saclay, le 17 juin 2024, par

Romain EGELE

Composition du jury

Membres du jury avec voix délibérative

Sarah COHEN-BOULAKIA Professeur, Université Paris-Saclay	Présidente
Massih-Reza AMINI Professeur, Université Grenoble Alpes	Rapporteur & Examinateur
Gavin CAWLEY Assistant professor équiv.HDR, University of East Anglia	Rapporteur & Examinateur
Gideon DROR Professor, Academic College of Tel Aviv	Examinateur
Claire MONTELEONI Professor, University of Colorado Boulder & Inria	Examinatrice

THESE DE DOCTORAT

NNT : 2024UPASG025

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre: Optimisation de Processus d'Apprentissage à Grande Échelle sur des Systèmes de Calcul Haute-Performance
Mots clés: Apprentissage Machine, Calcul Haute Performance, Optimisation des Hyperparamètres, Recherche d'Architectures de Réseaux de Neurones Artificiels, Optimisation Bayésienne

Résumé: Au cours de la dernière décennie, l'apprentissage automatique a connu une croissance exponentielle, portée par l'abondance de jeux de données, les avancées algorithmiques et l'augmentation de la puissance de calcul. Simultanément, le calcul haute performance (HPC) a évolué pour répondre à la demande croissante en calculs, offrant des ressources pour relever des défis scientifiques complexes. Cependant, le développement de processus d'apprentissage est souvent une suite d'essai-erreur basée sur des heuristiques, ce qui le rend difficile à suivre et chronophage. Les processus d'apprentissage machines sont construits à partir de modules qui offrent de nombreux paramètres configurables, des politiques d'augmentation des données, aux procédures d'entraînement et aux architectures de modèles. Cette thèse se concentre sur l'optimisation des hyperparamètres des processus d'apprentissage sur des systèmes HPC, tels que Polaris à Argonne National Laboratory. Les principales contributions comprennent : (1) l'optimisation Bayésienne parallèle décentralisée et asynchrone, (2) son extension à plusieurs objectifs, (3) l'intégration de methodes de rejet précoce et (4) la quantification de l'incertitude des réseaux de neurones profonds. De plus, un logiciel en libre acces, DeepHyper, est fourni, encapsulant les algorithmes proposés pour faciliter la recherche et l'application. La thèse met en évidence l'importance des méthodes d'optimisation Bayésienne des hyperparamètres pour les processus d'apprentissage, cruciales pour exploiter efficacement les vastes ressources de calcul des systèmes HPC modernes.

Title: Optimization of Learning Workflows at Large Scale on High-Performance Computing Systems
Keywords: Machine Learning,High-Performance Computing,Hyperparameter Optimization, Neural Architecture Search,Bayesian Optimization

Abstract: In the past decade, machine learning has experienced exponential growth, propelled by abundant datasets, algorithmic advancements, and increased computational power. Simultaneously, high-performance computing (HPC) has evolved to meet rising computational demands, offering resources to tackle complex scientific challenges. However, the development of learning workflows is often a sequential trial-error process based on heuristics, making it hard to track and time-consuming. Learning workflows are built from modules offering numerous configurable parameters, also known as hyperparameters, from data augmentation policies to training procedures and model architectures. This thesis focuses on the hyperparameter optimization of machine learning workflows on large-scale HPC systems, such as the Polaris supercomputer at the Argonne Leadership Computing Facility. Key contributions include (1) asynchronous decentralized parallel Bayesian optimization, (2) extension to multi-objective, (3) integration of early discarding, and (4) uncertainty quantification of deep neural networks. Furthermore, an open-source software, DeepHyper, is provided, encapsulating the proposed algorithms to facilitate research and application. The thesis highlights the importance of scalable Bayesian optimization methods for the hyperparameter optimization of learning workflows, which is crucial for effectively harnessing the vast computational resources of modern HPC systems.

Acknowledgements

The work presented in this thesis has been possible thanks to the support of many people. I would like to express my gratitude to all of them.

First, I would like to thank my advisors, Prof. Isabelle Guyon and Dr. Prasanna Balaprakash, for their guidance, support, and encouragement throughout my Ph.D. studies. I am grateful for their patience, their insightful comments, and their continuous support. I have learned a lot from them, and I am very thankful for the opportunity to work with them.

I would like to thank the members of my thesis committee, Prof. Sarah Cohen-Boulakia, Prof. Massih-Reza Amini, Prof. Gavin Cawley, Prof. Gideon Dror, and Prof. Claire Monteleoni for their time and their valuable feedback.

I would like to thank all my co-authors, acknowledged in each chapter of this thesis, for their collaboration and their contributions to this research. I am grateful for the opportunity to work with them, and I have learned a lot from them.

I would like to thank the members of the datascience team at the Argonne Leadership Computing Facility for their invuluable help in setting up machine learning software on HPC systems.

I would like to thank the members of the TAU team at the LISN, Université Paris-Saclay for their help and their support.

Last but not least, I would like to thank my wife, my family and my friends for their love, their support, and their encouragement. I am very grateful for their presence in my life.

Contents

1 - Introduction and contributions

1.1 . Introduction

Since 2010, the field of machine learning has experienced tremendous growth in both research and applications (Pugliese et al., 2021). This growth can be explained by the availability of large datasets, the development of new algorithms, and the increase in computational power. In particular, the availability of graphical processing units (GPUs) has helped to train large deep neural networks (DNNs) in a reasonable amount of time which has paved the way for many new applications. In parallel, the high-performance computing (HPC) community has been developing new systems to address the growing demand for computational power. These systems are now available to machine learning practitioners and can be leveraged to solve challenging science applications. However, the optimization of such learning workflows is still a challenge due to the large configurability of DNNs and the high computational cost of evaluating these configurations.

A machine learning workflow is a program that executes a sequence of operations to extract knowledge from data. It takes as *input* data corresponding to a specific task (*e.g.*, supervised learning) and produces as *output* a trained model. The model can then be used for different types of inferences such as prediction (and therefore called a predictor) or generation (and therefore called a generator). The modules of learning workflows can perform (1) data transformations (*e.g.*, augmentation, cleaning), (2) model training, (3) model evaluation and selection. Also, each module generally offers a large set of parameters to configure such as: (1) select the data augmentation policy which can include the type of transformations, their rate, and amplitude; (2) select the deep neural architecture which can include the number of layers, the type of activation functions, the regularization, the type of connections, etc.; (3) the metrics to evaluate and the selection protocol (*e.g.*, early stopping, cross-validation). A High-Performance Computing (HPC) system also known as a supercomputer is a collection of compute nodes connected through a high-speed network. Each node can be composed of one or more CPUs or "accelerators" (*e.g.*, GPUs). The nodes are connected to a shared storage system and memory as well as local. The HPC system can be used to accelerate the training of learning workflows or evaluate many in parallel.

The optimization of learning workflows has often been studied in the context of single-node systems or small clusters (*i.e.*, a few dozen parallel processes). For example and in chronological order, Bergstra et al. (2011) use 5 parallel process, Snoek et al. (2012) use 10 parallel processes, Falkner et al. (2018) use 32 parallel processes, Cho et al. (2020) use 6 parallel processes, Awad et al. (2021) use 64 parallel processes. Few works covered larger scales such as Snoek et al. (2015) which use a dynamic number of parallel processes between 300 and 800. Li et al. (2020) scale to 500 parallel processes but this is through a (cheap in overhead) random search combined with an early discarding strategy. Wang et al. (2018) scale Bayesian optimization to 500 parallel processes but mention a drop of

efficiency due to synchronization. Balaprakash et al. (2018b) scale Bayesian optimization to 1,024 parallel processes and it is one of the rare works that provide clear profiling of workers' activity. The profiling is presented through what is referred to as worker utilization, and it demonstrates a clear bottleneck on the scaling of Bayesian optimization when parallelized with the constant-liar strategy (Ginsbourger et al., 2010b).

All in all, the current literature makes it challenging to understand the benefit of parallelism and scalability for the optimization of learning workflows. This research will mostly experiment on the Polaris system at the Argonne Leadership Computing Facility which has 480 nodes in the production queue. Each node is equipped with 32 cores AMD EPYC "Milan" CPU and 4 NVIDIA A100 GPUs. This corresponds to a significantly larger scale than previously cited works with a total of 15,360 parallel CPU cores and 1,920 parallel GPUs. Polaris is a petascale supercomputer that serves as a developing platform for the Aurora exascale supercomputer. Aurora will be composed of 10,624 nodes. Each node will be equipped with 2 Intel Xeon CPU Max Series and 6 Intel Data Center GPU Max Series. Thus, it will provide a much larger computing scale than Polaris.

The goal of this thesis is **to develop and evaluate scalable methods for the optimization of learning workflows on large-scale high-performance computing systems**. The research spans various domains from machine learning and high-performance computing, to the application of these techniques to cancer research, geophysical modeling, and storage services. The theoretical aspect of this thesis intersects multiple theories such as probability and statistics, optimization, parallel computation, and learning theory, where the challenge has been to have a formal universal notation We provide a listing of our notations in Appendix A.

Early on and due to some properties of the optimization problem (*e.g.*, expensive evaluations, mixed-integer problem), the research focused on the class of model-based Bayesian optimization (BO) algorithms. Bayesian optimization is known to be challenging to scale due to multiple bottlenecks (Snoek et al., 2015; Balaprakash et al., 2018b). However, it is also known to have faster convergence (De Freitas et al., 2012), and as learning workflows are known to be expensive to evaluate this made Bayesian optimization a natural choice. The methodology mainly consists of relying on a probabilistic surrogate model (*e.g.*, Gaussian Process) to guide the optimization process instead of directly querying the real problem.

1.2 . Contributions

This thesis proposes a comprehensive framework for the scalable optimization of learning workflows on large-scale high-performance computing (HPC) systems. It primarily focuses on advancing Bayesian optimization techniques for efficient and scalable parallel hyperparameter optimization of deep neural networks (DNNs). The key contributions of the thesis can be summarized as follows:

1. **Scalable Parallel Bayesian Optimization** (Chapter 3): Introduces a decentralized and asynchronous approach to scale Bayesian optimization for hyperparameter

tuning across thousands of CPU and GPU processes, enabling optimization at scales previously unattainable (Egelé et al., 2023).

2. **Multi-objective Optimization** (Chapter 4): Extends Bayesian optimization to efficiently handle multi-objective scenarios through novel methods for objective normalization and constraints integration, enhancing its utility in practical problems (Égelé et al., 2023a).

3. **Early Discarding Techniques** (Chapter 5): Explores strategies to reduce computational demands of DNN training within hyperparameter optimization, demonstrating that a fixed number of training epochs can significantly reduce computational requirements (Égelé et al., 2023b).

4. **Uncertainty Quantification** (Chapter 6): Employs hyperparameter optimization for the quantification of uncertainty in DNN predictions, improving the reliability and interpretability of machine learning models (Égelé et al., 2021; Égelé et al., 2022; Maulik et al., 2023).

5. **DeepHyper Software Package**[1]: Contributes an open-source software package that encapsulates the developed algorithms for scalable optimization, providing a framework for asynchronous exploration and evaluation on HPC resources (Balaprakash et al., 2018a).

Figure 1.1 provides an overview of the software architecture of DeepHyper. The inputs defining the problem include the objective function f, which can return one or multiple objectives, and the hyperparameter search space Θ. Subsequently, a $\mathcal{S}$earch object represents any algorithm capable of exploring the hyperparameter search space, such as Bayesian optimization, random search, grid search, or genetic algorithms. This $\mathcal{S}$earch algorithm submits hyperparameter configurations to be evaluated by the Evaluator.

The Evaluator then executes the objective function f with the submitted hyperparameter configurations on available computing resources (referred to as workers). Upon completion of the evaluation of the objective function f, the search mechanism gathers the results. Both the submission and gathering processes support asynchronous programming, enabling non-blocking execution.

In the diagram, a blue rectangle represents a compute node, which serves as an atomic unit within HPC systems. Additionally, the purple background denotes the available "shared memory," which can be any mechanism facilitating data sharing among different processes (*e.g.*, a database, a filesystem).

1.3 . French Summary (Synthèse en Français)

[1]DeepHyper (Documentation: `deephyper.readthedocs.io` and Github: `github.com/deephyper/deephyper`, accessed March 2024)

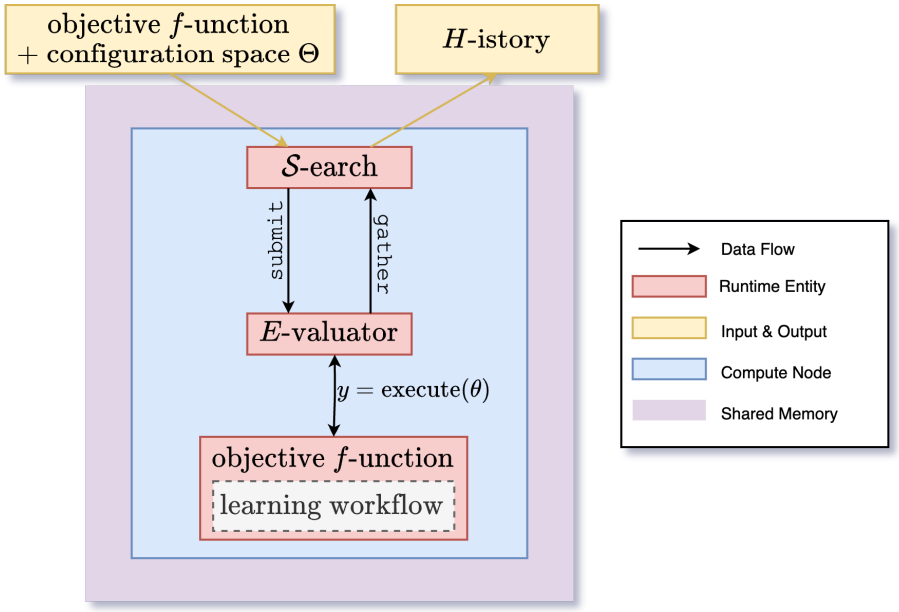


Figure 1.1: Overview of the software architecture of DeepHyper for the parallel optimization of learning workflows.

Au cours de la dernière décennie, les processus d'apprentissage automatique offrant connu une expansion exponentielle, stimulée par l'abondance de données disponibles, les progrès algorithmiques ainsi que l'augmentation de la puissance de calcul. En même temps, le calcul haute performance (HPC) a continué d'évolué pour répondre cette demande accrue en ressources de calcul, ouvrant ainsi la possibilité pour les processus d'apprentissage d'assimiler de plus grand volumes de données. Toutefois, les processus d'apprentissage machine sont souvent séquentiels, ce qui complique leur adaptation aux systèmes HPC, naturellement orientés vers le parallélisme massif.

Les processus d'apprentissage machine s'appuient sur des modules dotés de nombreux paramètres configurables, ainsi appelés "hyperparamètres" et souvent figés pour des raisons pratiques lors de l'apprentissage, allant des politiques d'augmentation des données, aux procédures d'entraînement et aux architectures des réseaux de neurones artificiels. L'optimisation des hyperparamètres de ces modules est essentielle pour développer des modèles performants, bien que celle-ci soit généralement très gourmande en calcul.

Cette thèse se concentre sur l'optimisation des hyperparamètres des processus d'apprentissage supervisé sur des systèmes HPC, comme Polaris à l'Argonne National Laboratory. Polaris dispose de 17 920 cœurs CPU et 2 240 accélérateurs GPU, offrant une plateforme de teste idéale pour l'expérimentation d'algorithmes d'optimisation des hyperparamètres à grande échelle. Les contributions majeures de cette thèse incluent:

1. **L'optimisation Bayésienne largement parallèle** (Chapitre 3) : une approche décentralisée et asynchrone permettant de paralléliser l'optimisation Bayésienne des

hyperparamètres pour l'apprentissage de réseaux de neurones profonds sur des milliers de processus CPU et GPU. Cette méthode améliore les résultats tout en réduisant significativement le temps de calcul nécessaire.

2. **L'optimisation multi-objectifs** (Chapitre 4) : une extension de l'optimisation Bayésienne parallèle pour gérer efficacement les scénarios où plusieurs objectifs doivent être optimisés simultanément. Cela permet par exemple de maximiser la qualité de l'apprentissage tout en minimisant le temps de réponse. Grâce à des méthodes novatrices de normalisation des objectifs et d'intégration de contraintes,la robustesse de l'optimisation est améliorées dans des problèmes pratiques (Égelé et al., 2023a).

3. **Les techniques de rejet précoce** (Chapitre 5) : alors que les contributions précédentes visent minimiser le nombre d'itérations de l'optimisation des hyperparamètres, cette approche vise à réduire les coûts de calcul des entraînements des réseaux de neurones pendant l'optimisation des hyperparamètres. L'objectif est de cesser l'entraînement des candidats peu prometteurs le plus tôt possible. Nous démontrons qu'un nombre fixe d'itérations d'entraînement peut réduire significativement et de façon prédictible les besoins en calcul tout en restant compétitif, voire supérieur à des méthodes plus complexes.

4. **La quantification de l'incertitude** (Chapitre 6) : cette partie montre comment l'optimisation des hyperparamètres peut être utilisée pour améliorer la quantification des incertitudes des prédictions des réseaux de neurones profonds, améliorant la fiabilité et l'interprétabilité des modèles d'apprentissage machines (Égelé et al., 2021; Égelé et al., 2022; Maulik et al., 2023). Notamment la recherche de réseaux de neurones avec des hyperparamètres variés permet de produire des prédictions plus diverses ce qui améliore significativement la quantification de l'incertitude.

5. **Le logiciel DeepHyper** : fournit une plateforme open source encapsulant les algorithmes développés pendant cette thèse, ce qui facilite la réutilisation et l'exploration scientifique des méthodes proposées.

Cette thèse souligne l'importance de l'optimisation Bayésienne des hyperparamètres pour les processus d'apprentissage, cruciale pour exploiter efficacement les vastes ressources de calcul des systèmes HPC modernes.

2 - Overview and formalism of the optimization of learning workflows

---

In this chapter, we present a thorough overview and formalism for optimizing learning workflows on high-performance computing systems. We start by introducing the notion of configurable learning workflows, providing clarity on their structure and components. Then, we delve into hyperparameter optimization, a pivotal aspect of this study, wherein we formally analyze the problem and scrutinize various key methods from the literature. Further details are provided on sequential Bayesian optimization, which serves as the cornerstone of this research. This includes a review of acquisition functions, the minimization of these functions, and the use of surrogate models. Additionally, we list performance metrics used for experimental assessments. Finally, the chapter introduces challenges associated with high-performance computing within the domain of optimizing learning workflows, thus laying the groundwork for more in-depth exploration in subsequent chapters.

2.1 . Formalism of configurable learning workflows

In this section, we introduce the concepts of *supervised learning workflows* that we will be optimizing throughout the remainder of this work.

In supervised learning, the aim is to construct a computer program (*i.e.*, a set of instructions executed by a computer) capable of generating predictions by leveraging a training dataset composed of examples of input-output pairs. The mechanism produces the datasets is later referred to as *supervisor*. The learning process involves two distinct types of programs: the *learner* and the *predictor*. The predictor is a program that executes the task of associating given inputs with corresponding outputs, serving as the manifestation of knowledge acquired by learning. On the contrary, the learner program is executes the task of assimilating this knowledge from the examples in the dataset, with the objective of identifying an optimal predictor. When the learner is made up of various subprograms (*a.k.a.*, modules), such as different preprocessing and modeling programs, it is often called a *learning workflow*. The learner and predictor usually take as inputs some fixed parameters that impacts their execution. This is mainly due to the practical consideration that computations are performed on finite resources (time and memory). We refer to such parameters as *hyperparameters*. Ultimately, hyperparameters are fixed parameters of the learning workflow.

Now, more formally, we follow the learning theory by Vapnik (1991) and borrow notations from Guyon et al. (2010); Liu (2021). The supervised learning process can be described through the following components:

- A *generator* of independent and identically distributed data X with support $\mathcal{X}$ (possibly a vector, a matrix or a tensor space) with $\mathrm{P}(X)$ its probability distribution, which is equivalent to $\mathrm{P}(x \in \mathcal{X})$ for any $x \in \mathcal{X}$.

- A *supervisor* which maps X to a target Y with support $\mathcal{Y}$ given a conditional distribution $\mathrm{P}(Y|X)$ also fixed and unknown, which is equivalent to $\mathrm{P}(y \in \mathcal{Y}|x \in \mathcal{X})$ for any $x, y \in \mathcal{X} \times \mathcal{Y}$.

- A *predictor*, is a function $\alpha_\theta : \mathcal{X} \to \mathcal{Y}$ from a space $\alpha_\theta \in \mathcal{A}_\theta$, θ is a hyperparameter vector. Then, $\mathcal{A}_\theta$ represents a set of functions that can be implemented for a fixed hyperparameter configuration θ of the predictor. This set generaly serves to estimate statistics of $\mathrm{P}(Y|X)$ (*e.g.*, mean, median, mode of the target).

- A *loss* function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ which describes the quality of the estimated statistic on the current task given a set of training data $D = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ where X_i, Y_i are $n \in [1, n]$, indepent and identically distributed (*i.i.d.*), random variables following the joint probability distribution $\mathrm{P}(X, Y)$. The probability of a dataset $D \in \mathcal{D}$ is denoted by $\mathrm{P}(D) = \mathrm{P}(X_1, Y_1, \ldots, X_n, Y_n)$.

In the case of parametrized predictors such as deep neural networks and to simplify our notations we write $\alpha_\theta(.)$ in place of $\alpha_\theta(.; w_\theta)$ where w_θ would be the parameters (*a.k.a.*, weights) of the neural network.

For some hyperparameters, let us define the *risk* which express the quality of a predictor on the learning task.

Definition 2.1 (Risk)**.** *Let $X, Y \sim \mathrm{P}(X, Y)$ be two random variables with support $\mathcal{X}, \mathcal{Y}$. Let $\alpha_\theta : \mathcal{X} \to \mathcal{Y}$ be a predictor configured with hyperparameters $\theta \in \Theta$. Let $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a loss function. The risk of α_θ according to the loss L on the supervised task defined by $\mathrm{P}(X, Y)$ is:*

$$R(\alpha_\theta) := \mathbb{E}_{X,Y}\left[L(Y, \alpha_\theta(X))\right] = \int_{X,Y} L(Y, \alpha_\theta(X)) d\mathrm{P}(X, Y)$$

Then, the learning problem is to find the predictor which minimizes the risk.

Problem 2.1 (Supervised Learning)**.** *Let X, Y be two random variables with support $\mathcal{X}, \mathcal{Y}$ respectively and joint probability distribution $\mathrm{P}(X, Y)$. Let $\alpha_\theta : \mathcal{X} \to \mathcal{Y}$ be a predictor configured with hyperparameters $\theta \in \Theta$. Let $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a loss function. The supervised learning problem is then to minimize the risk of α_θ according to the loss L on the supervised task defined by $\mathrm{P}(X, Y)$:*

$$\alpha_\theta^* = \underset{\alpha_\theta \in \mathcal{A}_\theta}{\arg\min}\, R(\alpha_\theta)$$

such that for all $\alpha_\theta \in \mathcal{A}_\theta$ we have $R(\alpha_\theta^*) \leq R(\alpha_\theta)$.

Our formulation does not explicitly account for the possibility of multiple solutions that minimize the risk, which often occurs for example because of problem symmetries.

Also, the only condition of optimality used is based on the value of the risk $R(\alpha_\theta^*) \leq R(\alpha_\theta)$. Due to the nature of the problems we study (*i.e.*, generally non-linear, non-continuous, non-convex and in later sections mixed-integer), very little can be said about other criteria of optimality (*e.g.*, first derivative tests such as Karush–Kuhn–Tucker conditions).

The discrepancy in the risk output space between solutions of Problem 2.1 and the actual data generator $\mathrm{P}(X, Y)$ is commonly known as *approximation error* (Barron, 1994). This is primarily attributed to the limited capacity of $\mathcal{A}_\theta$ to encompass a predictor that perfectly aligns with the actual data generator.

Returning to the assessment of risk, in practical scenarios, the joint distribution $\mathrm{P}(X, Y) = \mathrm{P}(Y|X)\,\mathrm{P}(X)$ is usually unknown. Therefore Problem 2.1 does not have a closed form solution. However, samples of the distributions may be given as training data, and therefore the problem shifts towards minimizing the *empirical risk* instead.

Definition 2.2 (Empirical Risk)**.** *Let $D = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ be a dataset of $(X_i, Y_i) \sim \mathrm{P}(X, Y)$, $i \in [1, n]$, i.i.d. random variables, with support $\mathcal{X} \times \mathcal{Y}$. Let $\alpha_\theta : \mathcal{X} \to \mathcal{Y}$ be a predictor configured with hyperparameters $\theta \in \Theta$. Let $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a loss function. The empirical risk of α_θ according to the loss L on the supervised task defined by $\mathrm{P}(X, Y)$ is:*

$$R_{emp}(\alpha_\theta) := \frac{1}{n} \sum_{i=1}^{n} L(Y_i, \alpha_\theta(X_i))$$

Since the empirical risk is determined by the dataset D, which is a random variable, it consequently becomes a random variable itself. The goal is now to minimize the empirical risk.

Problem 2.2 (Empirical Supervised Learning). *Let $D = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ be a dataset of $(X_i, Y_i) \sim \mathrm{P}(X, Y)$, $i \in [1, n]$, i.i.d. random variables, with support $\mathcal{X} \times \mathcal{Y}$. Let $\alpha_\theta : \mathcal{X} \to \mathcal{Y}$ be a predictor configured with hyperparameters $\theta \in \Theta$. Let $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a loss function. The supervised empirical learning problem is then to minimize the empirical risk of α_θ according to the loss L on the supervised task defined by $\mathrm{P}(X, Y)$:*

$$A_\theta^* = \underset{\alpha_\theta \in \mathcal{A}_\theta}{\arg\min}\ R_{emp}(\alpha_\theta)$$

where A_θ^* is a random variable, such that for all $\alpha_\theta \in \mathcal{A}_\theta$, $R_{emp}(A_\theta^*) \leq R_{emp}(\alpha_\theta)$.

Generally, the global optimality conditions for Problems 2.1 and 2.2 cannot be theoretically confirmed; they can only be empirically validated for solution quality. Furthermore, the difference in the risk output space between solutions of Problems 2.1 and 2.2 is commonly referred to as *estimation error* (Barron, 1994) (Figure 2.1). Since the resulting predictor A_θ^* is determined by the empirical risk, which is a random variable, it consequently becomes a random variable itself.

Let's define a deterministic learner, which can also be configured through its hyperparameters, as a function (or algorithm) that provides an estimated solution to Problem 2.2.

Definition 2.3 (Deterministic Learner). *Let $D = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ be a dataset of $(X_i, Y_i) \sim \mathrm{P}(X, Y)$, $i \in [1, n]$, i.i.d. random variables, with support $\mathcal{X} \times \mathcal{Y}$. Let $\alpha_\theta : \mathcal{X} \to \mathcal{Y}$ be a predictor configured with hyperparameters $\theta \in \Theta$. Let $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a loss function. Then, a configurable learner is a function $\beta_\theta : \mathcal{D} \to \mathcal{A}_\theta$ that, for a given configuration of hyperparameters $\theta \in \Theta$, and a training dataset $D \in \mathcal{D}$, returns an estimated optimal predictor $A_\theta = \beta_\theta(D)$ approximate solution to Problem 2.2. A_θ is a random variable because of randomness from D.*

The difference in the risk output space between Problem 2.2 and the expected predictor returned by a deterministic learner $\mathbb{E}_D\left[R_{\text{emp}}(A_\theta)\right]$ will be referred to as *learner error* (sometimes also referred to as the bias of the learner (Domingos, 2000)).

Definition 2.3 corresponds to a deterministic learner, even though we have presented the output predictor as a random variable. This means that for the same dataset, the returned predictor will provide the same outputs. For instance, consider the algorithm computing the least-square solution in linear regression, where the hyperparameter $\theta \in [1, 3]$ configures the number of polynomial features generated during data preprocessing.

However, in many cases, a learner may instead be a randomized algorithm. For instance, in the case of deep neural networks, randomness can arise from the initialization of random weights or the shuffling of the dataset during stochastic gradient descent (Bottou, 2010). In Random Forests (Breiman, 2001), randomness may stem from bootstrapping when constructing each tree (also known as bagging (Breiman, 1996)), random splits, or feature selection at each node of a tree. Hence, the outcome A_θ of a learner may also be influenced by such random effects.

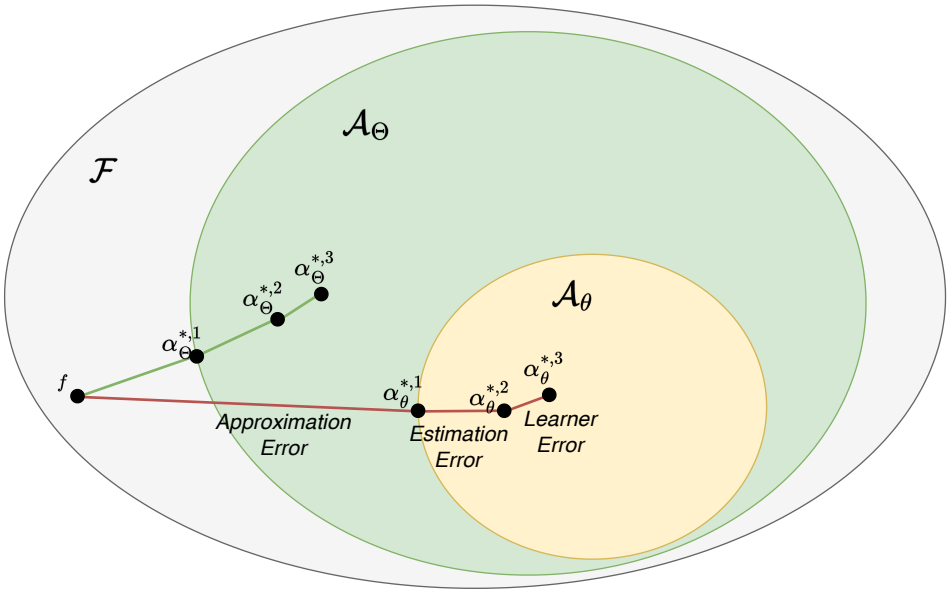


Figure 2.1: A visual representation of the hyperparameter optimization problem. $\mathcal{F}$ is the set of computable (*i.e.*, programmable) functions, and f is such a function that we wish to approximate. $\mathcal{A}_\theta$ (in yellow) is a possible set of predictors that corresponds to a fixed hyperparameter configuration θ. For example, it can corresponds to all weights that a deep neural network with a fixed architecture can take. $\mathcal{A}_\Theta$ (in green) is the union set of all $\mathcal{A}_\theta$ for all possible hyperparameter configurations $\theta \in \Theta$. The hyperparameter optimization process expores $\mathcal{A}_\Theta$ by sampling $\mathcal{A}_\theta$ subsets.

Definition 2.4 (Randomized Learner)**.** *Let $D = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ be a dataset of $(X_i, Y_i) \sim \mathrm{P}(X, Y)$, $i \in [1, n]$, i.i.d. random variables, with support $\mathcal{X} \times \mathcal{Y}$. Let $\alpha_\theta : \mathcal{X} \to \mathcal{Y}$ be a predictor configured with hyperparameters $\theta \in \Theta$. Let $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a loss function. Then, a randomized learner is a function $\beta_\theta : \mathcal{D} \times \mathbb{R} \to \mathcal{A}_\theta$ that, for a given configuration of hyperparameters $\theta \in \Theta$, and a training dataset $D \in \mathcal{D}$, returns an estimated optimal predictor $A_\theta = \beta_\theta(D, \mathcal{E})$ solution to Problem 2.2 and $\mathcal{E} \sim \mathrm{P}(\mathcal{E})$ with $\mathcal{E}$ encoding the cumulative random effects inherent to the learner β. A_θ is a random variable because of randomness from D and $\mathcal{E}$.*

Therefore, there are two sources of randomness, which may potentially interact, as seen in algorithms like stochastic gradient descent (where the ordering and value of samples introduce variability). One approach to assessing the effectiveness of a randomized learner is to choose one with low empirical risk on average, denoted as $\mathbb{E}_\mathcal{E}[A_\theta]$.

Now that we have defined the two main entities, the *predictor* and the *learner*, let's delve into further details about the hyperparameter space.

The hyperparameter space Θ encompasses all possible combinations of decisions that can configure the *predictor* and *learner*. Variables within this space can represent various aspects such as data-centric decisions (*e.g.*, preprocessing, augmentation, clean-

ing, normalization), learning procedure decisions (*e.g.*, optimization procedure of a neural network, learning rate, batch size, loss function), neural network architecture decisions (*e.g.*, types of neural network layers, connectivity between layers), and computational decisions (*e.g.*, parallelism configuration, memory limitations, optimized compute kernels).

Thus, the hyperparameter space can be described as a Cartesian product of different variable types:

- *Real*: a subset of real numbers $\mathbb{R}$ with corresponding order relation and distance (generally Euclidean).
- *Discrete*: a subset of natural numbers $\mathbb{Z}$ with a corresponding order relation and distance (generally Euclidean).
- *Categorical*: a set of values $\mathbb{K}$ with (Ordinal) or without (Nominal) order relation and no defined distance.

It can be represented in various formats such as a vector, a tree, or a graph. In this work, we primarily use vector representations in our formalism, although other data structures can be employed during implementation.

A space composed of such heterogeneous dimension types is termed as *mixed* and is known to pose challenges in optimization due to potential high non-regularity (*e.g.*, discontinuity in input and output spaces, frequent flatness/stationarity of the objective landscape). Additionally, it's common for some (child) hyperparameters to be valid only if a (parent) hyperparameter takes certain values. In such cases, there may exist redundancy (known as symmetries) as different hyperparameter vectors could represent the same learning workflow. Redundancy, where unique representation is lacking, can adversely affect the performance of statistical estimators.

To address this issue, we propose enforcing the representation of "invalid" hyperparameters with a predetermined value. Specifically, we set this predetermined value as the "lower bound" of the current hyperparameter dimension, thus breaking symmetries. In the case of nominal categorical hyperparameters, it will be a constant value from the categorical set. For example, if we have a hyperparameter representing the number of layers in a neural network, and an hyperparameter for each activation function of each of these layers, we can set the activation function of all inactive layers to a constant value (*e.g.*, "`identity`"). This approach ensures that the hyperparameter space is well-defined and unique (*i.e.*, no duplicate representation), thereby facilitating the optimization process.

In this section, we have outlined the fundamental problem in learning, which involves identifying an effective predictor for a predetermined set of hyperparameters.

Additionally, we have highlighted two distinct sources of randomness. The first pertains to the data and is represented through the random variables (X_i, Y_i). The second source of randomness is associated with the learner, which encompasses the model or algorithm utilized to tackle the learning problem, and is captured by the random variable $\mathcal{E}$.

In the subsequent section, we will delve into problems and methodologies concerning the exploration of hyperparameters for such predictors and learners.

2.2 . Hyperparameter optimization of learning workflows

Oftentimes implementations of algorithms (*a.k.a.*, programs) provide configurable parameters that impact the behavior of the algorithm during its execution. The problem of finding the parameters that optimize the performance of an algorithm is generally called "*algorithm configuration*" (AC).

A specific class of algorithms, presented in Section 2.1, is the class of machine learning (ML) algorithms. The parameters of ML algorithms that cannot be inferred during the *learning* phase (Problem 2.2) are named *hyperparameters*. Therefore, the hyperparameter optimization (HPO) problem is a subset of the algorithm configuration problem that focuses on machine learning algorithms. The HPO problem is now of great importance as ML algorithms have become extremely popular and the "performance" of these algorithms is sensitive to the configured hyperparameters.

However, while choosing the correct hyperparameters is important it is also notoriously difficult to select them properly, even for experts. This difficulty can be attributed to the quantity of available ML algorithms and hyperparameters, the lack of theoretical understanding about how such hyperparameters impact the performance (often resorting to heuristic decisions), and the computational requirements it necessitates to test.

Therefore, the research community has actively proposed automated procedures to resolve such challenges. In this chapter, we first present the formal problems (Section 2.2.1) that model such challenges, then we review methods (Section 2.2.3) from the literature that solves these problems.

2.2.1 . Formalism of the problem

In this section, we build on top of the previously introduced *predictor* and *learner*, from Section 2.1, to develop the hyperparameter optimization problems. Our formalism only considers the hyperparameter optimization problem without loss of generality to the combined algorithm selection and hyperparameter optimization (CASH) problem or the neural architecture search (NAS) problem (Eggensperger et al., 2013; Kotthoff et al., 2017; Thornton et al., 2013; Zela et al., 2018). Typically, for practical purposes, hyperparameter optimization in empirical supervised learning is formulated as a bi-level optimization problem (Guyon et al., 2010, Section 3).

Problem 2.3 (Hyperparameter Optimization of Empirical Supervised Learning)**.** *Let* $D = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ *be a dataset of* $(X_i, Y_i) \sim \mathrm{P}(X, Y)$*,* $i \in [1, n]$*, i.i.d. random variables, with support* $\mathcal{X} \times \mathcal{Y}$*. Let* $\alpha_\theta : \mathcal{X} \to \mathcal{Y}$ *be a predictor configured with hyperparameters* $\theta \in \Theta$*. Let* $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ *be a loss function. The hyperparameter optimization of empirical supervised learning problem is then a two-level problem where the goal is to find* θ *such that the empirical risk of its corresponding predictor* α_θ *is minimal according to the loss* L *on the supervised task defined given by* $\mathrm{P}(X, Y)$*:*

$$\Theta^* = \arg\min_{\theta \in \Theta} R_{emp}\left(A^*_\theta\right)$$
$$\text{s.t. } A^*_\theta = \arg\min_{\alpha_\theta \in \mathcal{A}_\theta} R_{emp}\left(\alpha_\theta\right)$$

where A^*_θ and Θ^* are random variables, such that for all $\theta \in \Theta$, for all $\alpha_\theta \in \mathcal{A}_\theta$, $R_{emp}\left(A^*_{\Theta^*}\right) \leq R_{emp}\left(\alpha_\theta\right)$.

Similar to supervised learning Problem 2.1, there may exist numerous optimal solutions for hyperparameters. Additionally, for each such optimal hyperparameter solution, there can be multiple optimal predictors (such as optimal weights). In practice, our goal is to identify just one combination of optimal solutions.

From the definition of the hyperparameter optimization problem, we can directly recognize the two-step process of the principle of structural risk minimization introduced by (Vapnik, 1991, Sections 7 and 8). In our scenario, the structure is defined by $S = \{\mathcal{A}_\theta : \forall \theta \in \Theta\}$ and may not necessarily be nested. This implies that two hyperparameter vectors can represent disjoint sets of predictors (for example, with different feature preprocessing).

In Figure 2.1, we present an illustration of the hyperparameter optimization process, where $f \in \mathcal{F}$ is a computable (or programmable) function (Sipser, 1996), and $\mathcal{A}_\Theta = \cup_{\theta \in \Theta} \mathcal{A}_\theta$ represents the set of all predictors that can be computed from any hyperparameter configuration $\theta \in \Theta$. The objective is to discover a better predictor by exploring subsets of $\mathcal{A}_\Theta$.

Subsequently, the resolution of the sub-problem is frequently handled by another algorithm, which is treated as a black box (meaning it is utilized based on observations of its inputs and outputs). In Section 2.1, we referred to such an algorithm as a learner and provided definitions for deterministic learners (see Definition 2.3) and randomized learners (see Definition 2.4). Therefore, let's directly focus on presenting the hyperparameter optimization of a randomized learner, as it represents the most general formalism.

Problem 2.4 (Hyperparameter Optimization of a Randomized Learner)**.** *Let $D = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ be a dataset of $(X_i, Y_i) \sim \mathrm{P}\left(X, Y\right)$, $i \in [1, n]$, i.i.d. random variables, with support $\mathcal{X} \times \mathcal{Y}$. Let $\alpha_\theta : \mathcal{X} \to \mathcal{Y}$ be a predictor configured with hyperparameters $\theta \in \Theta$. Let $L : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ be a loss function. Let $\beta_\theta : \mathcal{D} \times \mathbb{R} \to \mathcal{A}_\theta$ be a randomized learner. The hyperparameter optimization of a randomized learner is then to find θ such that the expected empirical risk of its output predictor α_θ is minimal according to the loss L on the supervised task defined given by $\mathrm{P}\left(X, Y\right)$*

$$\Theta^* = \arg\min_{\theta \in \Theta} \mathbb{E}_{\mathcal{E}}\left[R_{emp}\left(\beta_\theta(D, \mathcal{E})\right)\right]$$

where Θ^* is a random variable, and such that for all $\theta \in \Theta$, $\mathbb{E}_{\mathcal{E}}\left[R_{emp}\left(\beta_{\Theta^*}(D, \mathcal{E})\right)\right] \leq \mathbb{E}_{\mathcal{E}}\left[R_{emp}\left(\beta_\theta(D, \mathcal{E})\right)\right]$.

Usually, Problem 2.4 is presented as black-box optimization problem (Bergstra et al., 2011) of the form:

$$\Theta^* = \arg\min_{\theta \in \Theta} f(\theta)$$

where $f : \Theta \to \mathbb{R}$ is a noisy function. In our case, we unfolded the optimized function $f(\theta) = R_{\text{emp}}\left(\beta_\theta(D, \mathcal{E})\right)$ to clarify the source of randomness from the dataset D, the learner $\mathcal{E}$, and also link the hyperparameter optimization to the learning problem 2.3 previously introduced.

2.2.2 . Generalization and overfitting

In the previous section, to simplify the introduction of the hyperparameter optimization problems we used only one dataset D. However, in general, excessively optimizing a learning workflow (*i.e.*, its parameters and hyperparameters) by using a single dataset leads to overfitting (*i.e.*, memorize observed samples, Ying (2019)). In our case, we are interested in optimizing deep neural network learning workflows for which "overfitting" can easily happen as they are universal approximators (Hornik et al., 1989).

To resolve this issue, a common practice in hyperparameter optimization and neural architecture search is to split the dataset into three exclusive subsets $D = D_{\text{train}} \cup D_{\text{valid}} \cup D_{\text{test}}$ called the training, validation, and test datasets respectively. The training dataset D_{train} is used by the learner $\beta_\theta(D_{\text{train}}, \mathcal{E})$ to produce a predictor (*e.g.*, learning the parameters of a deep neural network), which corresponds to the lower level in Problem 2.3. The validation dataset D_{valid} is used to select the optimal learned predictor and its corresponding hyperparameters, which corresponds to the upper level in Problem 2.3. Finally, the test dataset D_{test} is used to report the final performance of the hyperparameter optimization algorithms and compare which one is better.

Even if this practice is common we wonder if repeated selection based on the validation dataset D_{valid} could lead to overfitting the validation dataset. This would imply that the empirical risk decreases (*i.e.*, improves) on the validation dataset but increases on the test datasets (*i.e.*, deteriorates). In the hyperparameter optimization setting, we are under a sequential adaptive process (*e.g.*, Bayesian optimization that we present in Section 2.2.4) that updates the selection of the best hyperparameters based on previously observed outcomes (*i.e.*, each new iteration depends on the outcome of past iterations). Therefore common generalization bounds, based on Hoeffding's inequality that assume independence between selected predictors, are not applicable (Blum and Hardt, 2015). Nevertheless, it was observed that over-adaptation of the learner on the validation dataset does not lead to overfitting the validation dataset (Recht et al., 2019; Blum and Hardt, 2015). Throughout our research, we also did not observe overfitting to the validation even when completing several thousands of optimization iterations.

We note that this methodology of a three-way splitting of the dataset and the issue of generalization of the selected best candidate is a general problem in machine learning. More specifically in the case of machine learning competitions (Pavão, 2023). In this case, training data are provided during the development phase which provides a leaderboard of competitors based on a hold-out dataset (corresponding to our validation dataset). Then, after possibly filtering the candidates (*e.g.*, to reduce noise in the selection) from this development phase leaderboard, a final phase is organized to select the winners (corresponding to our test dataset).

2.2.3 . Review of hyperparameter optimization methods

In this section, we conduct a literature review on hyperparameter optimization (HPO) and neural architecture search (NAS), guided by the taxonomy illustrated in Figure 2.2 and proposed by Elsken et al. (2019). This taxonomy categorizes approaches based on the *optimization strategy* (*a.k.a.*, search strategy) and the *performance estimation strategy*. The optimization strategy corresponds to the exploration of the hyperparameter space, denoted as Θ, while the performance estimation strategy corresponds to the methodology employed to solve the sub-problem outlined in Problem 2.3, namely, how the learning workflow is assessed.

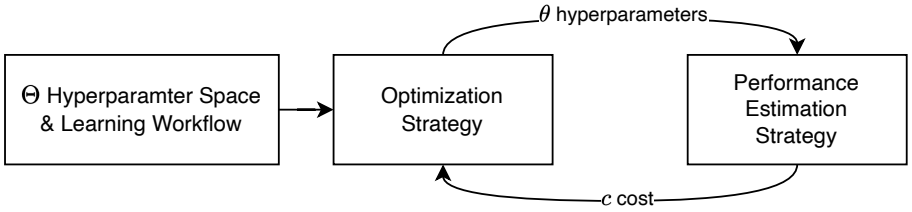
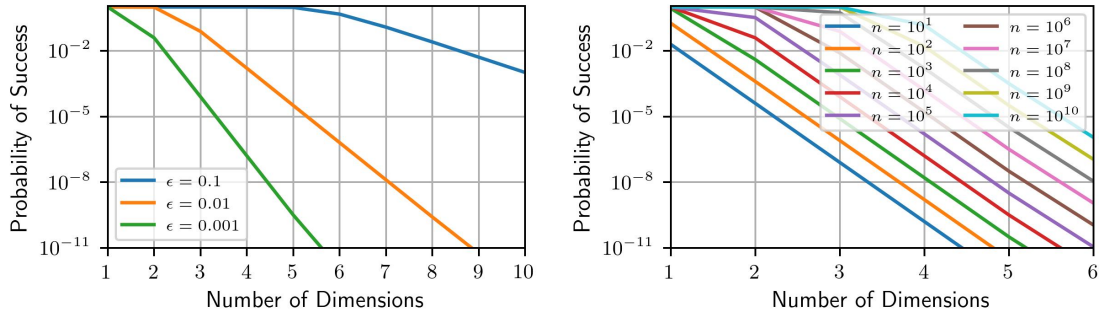


Figure 2.2: Components of hyperparameter optimization methods (Elsken et al., 2019).

The case of random hyperparameter optimization

Before introducing more complex strategies, we present random optimization, meaning sampling randomly (sometimes with an enforced prior or distribution) hyperparameters θ to test. From an HPC perspective, this strategy has the advantage of being stateless, with negligible overhead, and does not require to communicate information when parallelized. Hence, it has excellent parallel scalability. From a hyperparameter optimization perspective, random search is often preferred over grid search as it is more robust to the varying importance of hyperparameters (Bergstra and Bengio, 2012). Let us see quickly what would be the probability of finding the optimal solution with such a strategy. This probability exponentially reduces with the number of dimensions. For a given function f with d dimensions we want to evaluate the probability of finding $\approx \theta^*$. Assuming that we sample the space θ from a uniform distribution $\mathrm{P}(\theta) := \mathcal{U}([a,b]^d)$, then the probability of sampling at ϵ-precision for one dimension is $p_\epsilon := \mathrm{P}(|\theta - \theta^*| \leq \epsilon) = \mathrm{P}(\theta^* - \epsilon \leq \theta \leq \theta^* + \epsilon) = \frac{2\epsilon}{b-a}$. Now considering the sampling for each dimension to be independent, the probability of sampling at ϵ-precision each dimension for d-dimensions is $p_d := p_\epsilon^d$. Finally, assuming that draws are independent for each new evaluation, the probability of sampling at ϵ-precision each dimension after n-draws is $p_n := 1 - (1 - p_d)^n$.

In Figure 2.3, we present the probability of success p_n to find a solution at ϵ-precision for increasing number of dimensions to illustrate (1) how it exponentially converges to 0 (*a.k.a.*, curse of dimensionality) and (2) to understand when it is efficient and therefore help quantify the difficulty of a problem. In Figure 2.3a, $n = 10^4$ and we change the precision of the solution from $\epsilon = 10^{-1}$, 10^{-2} to 10^{-3}. As can be observed, increasing the



(a) Increasing Precision of the Solution (b) Increasing the Number of Samples

Figure 2.3: Probability of Success with Random Optimization

precision of the solution of a few digits makes the problem a lot harder for random optimization. Then, in Figure 2.3b the precision is fixed to $\epsilon = 10^{-3}$ and we vary the number of samples from 10 to 10^{10}. This shows the exponential increase of samples required to solve problems with more dimensions. In conclusion, if the problem becomes exponentially harder, even a perfect parallel scaling of random optimization could only increase the number of samples linearly (*i.e.*, number of samples per process $\times$ number of parallel processes) and will therefore not be able to close the gap.

Overview and synthesis of optimization strategies

In this section, we provide a concise overview of optimization strategies proposed in the hyperparameter optimization literature (Yang and Shami, 2020) and neural architecture search literature (Elsken et al., 2019). We categorize these methods based on the level of "information" they utilize from the learning workflow, specifically from the learner or predictor.

The first category of methods, termed "derivative-free" optimization (Larson et al., 2019), treats Problem 2.4 as pure black-box problems. In this scenario, the only available information comprises input-output pairs, where the input represents a hyperparameter configuration θ, and the output corresponds to a metric associated with the empirical risk (Definition 2.2). This class of methods proves highly practical, as it does not necessitate the examination or alteration of the internal workings of the learning workflow. Additionally, it readily lends itself to optimizing hyperparameters concerning data preprocessing, learning strategies, or neural architecture. Popular methods falling within this category include grid search (*a.k.a.*, factorial experimental design), random search (*a.k.a.*, random design), evolutionary algorithms (*e.g.*, genetic algorithms and swarm optimization), Covariance Matrix Adaptation, Bayesian optimization, Powell's methods, Nelder-Mead method, Simulated annealing, and Stochastic optimization.

The second class of methods involves derivatives of orders greater than 0 (*i.e.*, first-order, second-order, etc.). In this scenario, more information can be gleaned from the function f. For first-order methods, such as gradient-based optimization, the gradient of

f with respect to the input configuration θ can be utilized. For second-order methods, the Hessian can be employed. In the realm of neural architecture search (NAS), optimization methods frequently leverage such information in a category of algorithms termed "differentiable architecture search" (DARTS), as introduced by (Liu et al., 2019b). This involves relaxing the optimization problem, making a non-differentiable discrete problem differentiable by mapping it to a problem that operates on continuous support. However, it is important to note that these methods generally incur a substantial memory cost and are often prone to instability (Li et al.; Zela et al.; Chu et al., 2021).

Overview and synthesis of performance estimation strategies

Having outlined optimization strategies capable of exploring the search space of learning workflows, we will now offer an overview of methods determining how a hyperparameter configuration θ should be evaluated. Evaluation strategies primarily seek to minimize the time needed to complete the optimization within a reasonable timeframe.

Many evaluation strategies have been proposed in the hyperparameter optimization and neural architecture search literature. The perfect solution aims to predict the cost of a given hyperparameter configuration with minimal time investment. Strategies can be categorized along three axes: (1) *parameter sharing*, (2) *one-shot*, and (3) *multi-fidelity*. In parameter sharing methods (Pham et al., 2018; Chu et al., 2021), particularly relevant to deep learning workflows, the concept involves reusing weights from a shared set to initialize the learning of each new candidate. One-shot methods (Guo et al., 2020; Elsken et al., 2019) seek to estimate the ranking of candidates without undertaking a full training procedure, which typically consumes a significant portion of the budget. Finally, multi-fidelity methods often employ a discretized concept of training budget (*e.g.*, training iterations). They dynamically allocate the training budget based on observed intermediate costs (Domhan et al., 2015; Li et al., 2017; Jamieson and Talwalkar, 2016), or do so statically (Égelé et al., 2023b). Generally, multi-fidelity methods are less intrusive and thus necessitate fewer adjustments to the inner logic of the learning workflow.

2.2.4 . More details on sequential Bayesian optimization

In this section, we provide more details about sequential Bayesian optimization, which serves as the fundamental algorithm underpinning our research. We chose Bayesian optimization as it is known to be efficient for the optimization of expensive black-box functions.

First, we introduce a generic template of a sequential Bayesian optimization algorithm. Subsequently, we examine three key components of this method: (1) the acquisition function (Section 13), (2) the resolution of the sub-problem aimed at minimizing the acquisition function (Section 13), and (3) the surrogate models (Section 13).

The sequential Bayesian optimization algorithm, also known as Efficient Global Optimization (EGO), was originally introduced by Jones et al. (1998). The core concept of EGO is to employ surrogate models fitted on a subset of evaluations from the actual cost or constraint functions, aiming to reduce the overall number of direct function evaluations,

which may be infeasible in certain cases (*e.g.*, due to unmanageable cumulative time). In Algorithm 1, we present a generic template of this methodology. After initializing the surrogate model (line 2), the algorithm enters the optimization loop (lines 3-12). At each iteration, a hyperparameter configuration θ is selected (lines 4-9). This selection can be achieved through randomized sampling (line 5), typically to gather initial samples, or via the updated surrogate model (lines 7-8). Subsequently, the hyperparameter configuration is evaluated on the actual function f (line 9). Upon completion of the evaluation, the suggested optimal hyperparameters are updated (line 10), for instance, by identifying the hyperparameters with the minimal observed cost if f is a deterministic function. This process iterates until certain stopping criteria are met, such as reaching the maximum number of evaluations, maximum time limit, or cost stagnation.

It is crucial to highlight that Algorithm 1 exclusively provides suggested optimal hyperparameters `thetaStar` and does not yield a trained predictor. In real-world scenarios, it is feasible to checkpoint trained predictors throughout the optimization process or conduct the training once optimization concludes, utilizing the suggested hyperparameters.

Algorithm 1: Bayesian Optimization (*a.k.a.*, Efficient Global Optimization (EGO))

Inputs : `thetaSpace`: a configuration space
`nInitial`: the number of initial hyperparameter configurations
`f`: a function that returns the cost of the learning workflow
Output: `thetaStar` the recommended hyperparameter configuration.

```
1  thetaArray, costArray ← New empty arrays of hyperparameter configurations and costs ;
2  model ← New surrogate model ;
3  while stopping criteria not valid do
4      if Length of thetaArray < nInitial then
5          theta ← Sample hyperparameter configuration from thetaSpace ;
6      else
7          Update model with thetaArray, costArray ;
8          theta ← Returns theta in thetaSpace  that minimizes the acquisition function for current model ;
9      end
10     cost ← Returns the cost of learning workflow f(theta) ;
11     thetaArray, costArray ← Concatenate thetaArray with [theta] and costArray with [cost];
12     thetaStar ← Update recommendation ;
13 end
```

Review of acquisition functions

In Bayesian optimization, as depicted in Algorithm 1, we utilize an *acquisition function* to assign a scalar score to candidate configurations, enabling their ranking and selection for suggestion. Formally, an acquisition function takes the form $a(\theta) = s \in \mathbb{R}$, typically based on the most recent updated model. Much of the Bayesian optimization literature operates under the assumption that the optimization process follows a Gaussian process (GP) framework, wherein $f(\theta) \sim \mathcal{N}(\mu(\theta), \sigma^2(\theta))$. Acquisition functions under this GP assumption fall into three main categories: (1) optimistic, (2) improvement-based, and (3) information-based. In the following, we outline the key steps for deriving some of the most commonly used acquisition functions, while omitting most of the technical details.

First, from the class of optimistic acquisition functions, the *lower confidence-bound* (LCB) (Cox and John, 1992) is given by:

$$a_{\text{LCB}}(\theta;\kappa) := \mu(\theta) - \kappa \cdot \sigma(\theta) \tag{2.1}$$

where κ is a parameter controlling the exploitation/exploration trade-off. The larger is κ the more is the exploration as we focus on configurations θ with uncertain outcomes. Also, $\kappa = 1.96$ corresponds to a 95% confidence interval in the predicted score. The LCB can be interpreted as "optimistic" as it considers the best possible estimated outcome.

Then, for the class of improvement-based acquisition functions, we need to start by defining a notion of *improvement* as:

$$I(\theta;\xi) := \max(f(\theta^{\circledast}) - f(\theta) - \xi, 0) \tag{2.2}$$

$$= \max(f(\theta^{\circledast}) - \mu(\theta) - z \cdot \sigma(\theta) - \xi, 0) \quad \text{with} \quad z \sim \mathcal{N}(0,1) \tag{2.3}$$

where $f(\theta^{\circledast})$ is the usually the best-observed score or current "recommendation". Therefore, Equation 2.2 quantifies the improvement of configuration θ concerning the current best score. The ξ parameters help to tune the magnitude improvements of interest. In Equation 2.3 the GP assumption is used to apply the "reparametrization trick" on $f(\theta)$.

From the definition of improvement we can now present the *probability of improvement* (PI) (Kushner, 1964) as:

$$\begin{aligned} a_{\text{PI}}(\theta;\xi) &:= \mathrm{P}\left(0 < I(\theta;\xi)\right) \\ &= \mathrm{P}\left(f(\theta) < f(\theta^{\circledast}) - \xi\right) \\ &= \mathrm{F}\left(f(\theta^{\circledast}) - \xi\right) \\ &= \Phi\left(\frac{f(\theta^{\circledast}) - \mu(\theta) - \xi}{\sigma(\theta)}\right) \end{aligned}$$

where $\mathrm{F}(.)$ is the cumulative distribution function of $f(\theta)$, and $\Phi(.)$ is the cumulative distribution function of the standard normal distribution (*i.e.*, mean zero and variance one).

Another improvement-based acquisition function is the *expected improvement* (EI) (Jones et al., 1998) which is more robust to noise than PI and is given by:

$$\begin{aligned} a_{\text{EI}}(\theta;\xi) &:= \mathbb{E}\left[I(\theta;\xi)\right] = \int \max(f(\theta^{\circledast}) - \mu(\theta) - z \cdot \sigma(\theta) - \xi, 0) \cdot \varphi(z)\, dz \\ &= \left(f(\theta^{\circledast}) - \mu(\theta) - \xi\right) \Phi\left(\frac{f(\theta^{\circledast}) - \mu(\theta) - \xi}{\sigma(\theta)}\right) + \sigma(\theta)\varphi\left(\frac{f(\theta^{\circledast}) - \mu(\theta) - \xi}{\sigma(\theta)}\right) \\ &= \left(f(\theta^{\circledast}) - \mu(\theta) - \xi\right) a_{\text{PI}}(\theta;\xi) + \sigma(\theta)\varphi\left(\frac{f(\theta^{\circledast}) - \mu(\theta) - \xi}{\sigma(\theta)}\right) \end{aligned}$$

where $\varphi(.)$ is the distribution function of the standard normal distribution.

Finally, the class of information-based acquisition functions, generally based on mutual information such as in *entropy search* (ES)(Hennig and Schuler, 2012) and *predictive entropy search* (PES)(Hernández-Lobato et al., 2014). The *mutual-information* acquisition function is given by:

$$a_{\text{MI}}(\theta) := \mathbb{I}\left(\{\theta, f(\theta)\}; \theta^{*} | D_t\right)$$

where $\mathbb{I}(.;.)$ evaluates the mutual information between $\{\theta, f(\theta)\}$ and θ^* given a set $D_t = \{(\theta_1, f(\theta_1)), \ldots, (\theta_t, f(\theta_t))\}$ of t observations. However, such acquisition function is often hard to estimate due to untractable integral functions and therefore resorting to Monte-Carlo estimation. A more tractable quantity is given by the *max-value entropy search* (MES) (Wang and Jegelka, 2017):

$$\alpha_{\text{MES}}(\theta) := \mathbb{I}(\{\theta, f(\theta)\}; f(\theta^*)|D_t)$$

where the mutual information is taken for the observed outcome instead.

Solving the sub-problem of minimizing the acquisition function

Another essential aspect of Bayesian optimization, as outlined in Algorithm 1, revolves around addressing the minimization problem of the acquisition function (line 8). This sub-problem's resolution presents a significant challenge and cannot be easily resolved. Various solvers may be employed for this task, depending on the surrogate model and acquisition function utilized, such as gradient-based methods, genetic algorithms, or random sampling. The complexity arises from the surrogate models' aim to closely approximate the target cost function, which entails capturing the optimization-related intricacies of this function, such as its non-convexity. Furthermore, depending on the surrogate model or acquisition function employed, derivability properties, and thus the existence of a gradient function, may or may not be present (*e.g.*, Random Forests). Moreover, striking a balance between the solution quality of the sub-problem and computational cost necessitates careful consideration. This minimization process can significantly increase the computational burden of the Bayesian optimization procedure, potentially rendering it less effective by diminishing the computational gap between the real function and the approximated sub-problem resolution.

Review of surrogate models

The final crucial component of Bayesian optimization, as depicted in Algorithm 1, is the surrogate model. As discussed in Section 13 regarding acquisition functions, the Bayesian optimization (BO) algorithm ranks candidate configurations by combining various statistics from the distribution $\mathrm{P}(f(\theta))$. Consequently, a surrogate model is necessary to estimate such quantities. Additionally, it is worth noting that we have assumed the BO process to be a Gaussian process, with the cost distribution $\mathrm{P}(f(\theta))$ described as a normal distribution denoted by its mean $\mu(\theta)$ and variance $\sigma^2(\theta)$. We now introduce two primary classes of surrogates for BO, both of which are intriguingly non-parametric models.

Gaussian processes models: Gaussian processes (GPs) (Rasmussen and Williams, 2006) are Bayesian models where we assume a prior distribution over possible functions and then update a posterior distribution based on observations. This class of surrogate models is often employed to impose regularity assumptions, such as smoothness, stationarity, and characteristic length-scale, on the target unknown function. Regularity is enforced through properties of the covariance function of the Gaussian process, which

is computed using a kernel function. Therefore, selecting the appropriate kernel function is crucial for the success of Gaussian processes. However, this introduces additional parameters to tune, which goes against the objective of hyperparameter optimization. It is possible to marginalize the hyperparameters of the Gaussian process to avoid the need for tuning them. Gaussian processes naturally handle continuous variables, but special treatment is required to deal with discrete and categorical variables. Additionally, Gaussian processes exhibit cubic complexity with respect to the number of observations, rendering them impractical for large-scale parallel optimization settings.

Random Forests models: Surrogate models based on tree ensembles, commonly referred to as forests, are widely utilized in Bayesian optimization for the hyperparameter tuning of learning workflows (Hutter et al., 2014b). In this approach, trees are constructed to minimize the expected squared error, typically achieved through a greedy or randomized optimization process that minimizes the estimated variance within the leaves. Such trees are employed to estimate the conditional expected outcome based on the input variables. The term "Random Forests" is often used interchangeably in the literature to describe various types of sets of randomized decision trees. In our study, we adhere to the terminologies outlined by Geurts et al. (2006) and highlight their respective characteristics:

- **Tree Bagging** (TB): an ensemble of bootstraped regression trees, with all features available at each split and the “best”-split rule (Breiman, 1996; James et al., 2013).
- **Random Subspace** (RS): an ensemble of regression trees, with $\sqrt{n}$ randomly sampled features available at each split and the “best”-split.
- **Random Forest** (RF): an ensemble of bootstraped regression trees, with $\sqrt{n}$ randomly sampled features available at each split and the “best”-split rule (Breiman, 2001).
- **Extremely Randomized Trees** (ET): an ensemble of regression trees, with all features available at each split and the “random”-split rule (Geurts et al., 2006).
- **Mondrian Forest** (MF): an ensemble of Mondrian regression trees (Lakshminarayanan, 2016; Lakshminarayanan et al., 2016), with all features available at each split.

we consider tree bagging, random subspace, random forest, and extremely randomized trees to be part of the class of **usual regression trees** as they follow the same generic algorithm but include different sources of randomness. In table 2.1 we provide a synthetic overview of sources of randomness present in these algorithms.

This clarification is important as depending on the machine learning framework (*e.g.*, Scikit-Learn) or hyperparameter optimization software (*e.g.*, SMAC3) the implementations corresponding to these models can vary (also within the same framework, by switching from classification to regression). For example, in SMAC3 (and the corresponding PyRFR package) what is denoted as Random-Forest is closer in spirit to Extremely randomized trees as it randomly samples the split of each node.

	Bootstrapping	Feature	Split
Tree Bagging (TB)	x		
Random Space (RS)		x	
Random Forest (RF)	x	x	
Extremely Randomized Trees (ET)			x
Mondrian Forest (MF)		x	x

Table 2.1: Randomness in Regression Ensemble of Trees Models

2.2.5 . Overview of metrics of performance

Having elucidated the core problems and existing methods, this section offers an overview of metrics utilized for quantitative assessments of learning workflow optimization. These metrics commonly feature in theoretical convergence proofs of Bayesian optimization (Srinivas et al., 2010; De Freitas et al., 2012; Astete-Morales et al., 2016). Also, it is interesting to note that at the moment common practices of benchmarking in optimization such as performance profiles (Dolan and Moré, 2002) are not yet employed in hyperparameter optimization.

The first metric, the regret (*a.k.a.*, instantaneous regret, or uniform rate), considers the quality of the last observed outcome.

Definition 2.5 (Regret)**.**

$$r(i) := f(\theta_i) - f(\theta^*)$$

where $i \in [|1, i_{max}|]$ *denotes the optimization steps with* $i_{max} \in \mathbb{N}^*$ *being the last or maximum step,* $f(\theta^*) = \min_{\theta \in \Theta} f(\theta)$ *is the true global optimum and* $f(\theta_i)$ *is the last observation at step* i*.*

The second metric, the simple regret, considers only the quality of the recommended outcome.

Definition 2.6 (Simple Regret)**.**

$$r^{\circledast}(i) := f(\theta_i{}^{\circledast}) - f(\theta^*)$$

where $i \in [|1, i_{max}|]$ *denotes the optimization steps with* $i_{max} \in \mathbb{N}^*$ *being the last or maximum step,* $f(\theta^*) = \min_{\theta \in \Theta} f(\theta)$ *is the true global optimum and* $\theta_i{}^{\circledast}$ *is the recommendation after observing* θ_i*.*

The third metric, cumulative regret, considers the quality of the full set of observed outcomes (*i.e.*, the complete optimization process).

Definition 2.7 (Cumulative Regret)**.**

$$R(i) := \sum_{i' \in [|1,i|]} r(i')$$

where $i \in [|1, i_{max}|]$ *denotes the optimization steps with* $i_{max} \in \mathbb{N}^*$ *being the last or maximum step, and* $r(i)$ *is the regret from Definition 2.5.*

The three metrics introduced in Definitions 2.5, 2.6 and 2.7 mainly correspond to the performance evaluation of sequential optimization processes such as Algorithm 1. These metrics assume that (1) all optimization steps (lines 4-9 in Algorithm 1) have uniform duration, and (2) all evaluation steps (line 10 in Algorithm 1) also have uniform duration. In our case, we often use time-related versions of these metrics that we now introduce. For simplicity, we use the same notations as the previously introduced metrics. In the case of discrete regrets, we will use "iterations" or "evaluations" as the name of the absciss in our graphics. Similarly, in the case of time-related regrets, we will use "time" as the name of the absciss in our graphics.

The first, the temporal regret, is the time-related version of the regret from Definition 2.5.

Definition 2.8 (Temporal Regret)**.**

$$r(t) := f(\theta_t) - f(\theta^*)$$

where $t \in [0, t_{max}]$ denotes the current time of the optimization process with $t_{max} \in \mathbb{R}^+$ being the maximum time, $f(\theta^*) = \min_{\theta \in \Theta} f(\theta)$ is the true optimum and $f(\theta_t)$ is the last observation at time t.

The second, the temporal simple regret, is the time-related version of the simple regret from Definition 2.8.

Definition 2.9 (Temporal Simple Regret)**.**

$$r^{\circledast}(t) := f(\theta_t{}^{\circledast}) - f(\theta^*)$$

where $t \in [0, t_{max}]$ denotes the current time of the optimization process with $t_{max} \in \mathbb{R}^+$ being the maximum time, $f(\theta^*) = \min_{\theta \in \Theta} f(\theta)$ is the true global optimum and $\theta_t{}^{\circledast}$ is the recommendation at time t.

The third, the temporal cumulative regret, is the time-related version of the cumulative regret from Definition 2.7.

Definition 2.10 (Temporal Cumulative Regret)**.**

$$R(t) := \int_0^t r(t')dt'$$

where $t \in [0, t_{max}]$ denotes the current time of the optimization process with $t_{max} \in \mathbb{R}^+$ being the maximum time, and $r(t)$ is the temporal regret from Definition 2.8.

This continuous version of the cumulative regret allows us to compare optimization processes producing different numbers of observations (*e.g.*, when using more parallel workers, or due to the time consumed by optimization steps). While the temporal cumulative regret is a natural time-related version of cumulative regret in practice we use the following formulation as we have a discrete set of observations:

$$R(t_i) \approx \sum_{i'=0}^{i} r(t_{i'}) \cdot (t_{i'+1} - t_{i'}) \tag{2.4}$$

where t_i are the observations times in increasing order, $t_0 = 0$ and $t_{i+1} = t_{\text{max}}$ by convention.

Last but not least, none of these metrics (either discrete or continuous) take into consideration the randomness from f. To resolve this issue it is often possible to simply replace the regret (Definitions 2.5 and 2.8) with the expected regret $\bar{r}(t) := \mathbb{E}\left[r(t)\right]$ (Astete-Morales et al., 2016).

2.3 . Opportunities and challenges of high-performance computing

In the preceding sections of this chapter, we introduced learning workflows and techniques for their optimization. We will now delve into the challenges entailed in optimizing learning workflows on high-performance computing (HPC) systems

2.3.1 . Introduction to high-performance computing

High-performance computing (HPC) systems typically comprise compute nodes, each containing multiple processing units and memory tiers. Traditionally, these units were solely CPUs, but it is now common to find a mix of processing units within a single node. For example, a node may incorporate a combination of CPUs, GPUs, and TPUs. These processing units, whether within a node or across nodes, can operate in parallel, offering both inter and intra-node parallelism.

For instance, multiple nodes can be employed concurrently, with each node harnessing multiple cores of a single CPU simultaneously. Within each core, parallel operations such as element-wise sums can be executed. Additionally, each node is furnished with various types of memory. Determining the optimal utilization of these heterogeneous resources, including processing units and memory, presents a significant challenge.

The primary HPC system employed in this research is Polaris[1], located at the Argonne Leadership Computing Facility. Each node of the Polaris system features 1 AMD Zen 3 (Milan) CPU with 32 cores, supplemented by 4 NVIDIA A100 GPUs. With a collective count of 560 nodes, Polaris boasts a total of 17,920 CPU cores and 2,240 GPUs. Additionally, each node is outfitted with DDR4 memory, a local SSD, and has access to a shared network filesystem.

2.3.2 . Opportunities of high-performance computing for machine learning

There is a clear and notable correlation between the advancement of machine learning algorithms (Sevilla et al., 2022) and the increasing capabilities of computation (Sevilla et al., 2022). This research is conducted in part at Argonne National Laboratory, a facility of the United States Department of Energy (DOE). As part of the Exascale Computing Project[2] (ECP), the DOE has initiated the construction of a series of exascale supercomputers, capable of achieving exaFLOPS computation speeds (10^{18}, or a billion billion calculations per

[1]Polaris Hardware Overview: https://docs.alcf.anl.gov/polaris/hardware-overview/machine-overview/ (accessed March 2024)

[2]Exascale Computing Project: https://www.exascaleproject.org/about/ (accessed March 2024)

second). Previous systems operated at the petascale level, reaching petaFLOPS computation capacities. In 2021, the first exascale supercomputer, Frontier[3], was unveiled at Oak Ridge National Laboratory, another DOE facility. Comprising 9,408 AMD compute nodes, 606,208 CPU cores, and 8,335,360 GPU cores, Frontier achieved a peak performance of 1.194 exaFLOPS (measured by Rmax, a specific metric for assessing computing system capabilities). Shortly following Frontier's launch, the Aurora[4] (at Argonne National Laboratory) and El Capitan[5] (Lawrence Livermore National Laboratory) supercomputers are set to be introduced. Polaris, the petascale system utilized in our research, serves as a testbed[6] platform to prepare for Aurora. As a result, this research is centered on developing optimization algorithms for learning workflows, aimed at efficiently leveraging these computing capabilities.

In our current context, our attention is directed towards optimizing the hyperparameters of a deep neural network. Parallelization within this framework involves two primary aspects: (1) identifying the optimal method for parallelizing the training of a single deep neural network, which may entail using one or multiple GPUs or employing one or multiple nodes, and (2) formulating strategies for parallelizing the assessment of multiple instances of deep neural network training, each with unique hyperparameter configurations.

2.3.3 . Challenges of optimizing learning workflows at large scale

Several challenges are associated with hyperparameter optimization of machine learning on high-performance computing (HPC) systems:

1. **Computational Resources**: Hyperparameter optimization often requires significant computational resources, including time, memory, processing power, and communication bandwidth. HPC systems can provide these resources, but efficiently managing them to accommodate the computational demands of hyperparameter optimization can be challenging. These costs justify our choice to build on top of Bayesian optimization methods that usually have faster convergence.

2. **Variability**: The use of computational resources of a learning workflow is variable. It usually partially depends on (1) the hyperparameter configurations and (2) the usage of shared computational resources (*e.g.*, filesystem). To illustrate the influence of the hyperparameter configuration, a deep neural network with fewer neurons per layer and fewer layers trains faster than a larger one. Such a simple example may represent the underlying issue as simple but as soon as the number and types of hyperparameters to configure increases it becomes extremely difficult to predict the computational cost of each evaluation. For example, imagine configuring the

[3]Frontier supercomputer: `https://www.olcf.ornl.gov/frontier` (accessed March 2024)
[4]Aurora supercomputer: `https://www.alcf.anl.gov/aurora` (accessed March 2024)
[5]El Capitan supercomputer: `https://asc.llnl.gov/exascale/el-capitan` (accessed March 2024)
[6]Argonne's 44-Petaflops "Polaris" Supercomputer Will Be Testbed for Aurora, Exascale Era (accessed March 2024)

type of training procedure, the stopping criteria of this training procedure, the size of each neural network layer, and the activation function of these layers. To illustrate the influence of the usage of shared computational resources, imagine thousands of deep neural network trainings loading simultaneously a large dataset from the filesystem to start their training, and later checkpointing their learned weights on this filesystem.

3. **Parallel Optimization**: Parallelizing an optimization process is often difficult as they often are fundamentally sequential processes. Simple parallelization can often result in stagnation of the process. For example, this is well-known for the parallelization of gradient-based optimization methods. Therefore, a core challenge will be to develop a parallel Bayesian optimization method that keeps improving with increased parallelism. Effectively parallelizing this algorithm on HPC systems requires careful consideration of communication overhead, load balancing, and resource utilization to ensure efficient utilization of computing resources (Wozniak et al., 2018).

4. **Algorithm Selection**: Choosing the appropriate hyperparameter optimization algorithm for the HPC environment is crucial. Different algorithms have varying computational requirements, convergence properties, and scalability characteristics, making it essential to select the most suitable algorithm for the specific application and computing infrastructure. Our goal will be to propose an algorithm that can adapt to any parallel scale and still provide the best outcome.

Last but not least, from a practical perspective, HPC systems often have extremely *variable hardware and software architectures across centers*. A few examples are the scheduler (*e.g.*, PBS, SLURM, COBALT), the CPU architecture (*e.g.*, x86, arm), the GPU architecture (*e.g.*, NVIDIA, AMD), the interfaces of the network, and the operating system can change. Therefore, to conduct our experiments and make our research reproducible we will have to develop parallel optimization methods that can generalize (easily) across such differences.

2.4 . Conclusion

In this chapter, we introduced the foundational aspects of this thesis, which encompasses learning workflows, their optimization, and the benefits of leveraging high-performance computing systems. The chapter started by developing the notion of learning workflows, specifically focusing on the class of supervised learning workflows. This section introduced the structure and learning dynamics of these workflows, emphasizing their configurability through their hyperparameters.

Then, we delved into the optimization of learning workflows from the viewpoint of hyperparameter optimization. We presented hyperparameter optimization as the most general problem that encompasses algorithm selection and neural architecture search.

Bayesian optimization was highlighted as a promising approach for hyperparameter optimization, given its improved convergence rate that reduces the number of evaluations required. However, the scalability of Bayesian optimization in parallel computing environments remains a significant challenge.

The third part of the chapter was dedicated to dissecting the opportunities and obstacles presented by high-performance computing (HPC) systems in the context of learning workflow optimization. We discussed how the vast computational resources available within HPC systems could be leveraged to accelerate the optimization process, albeit with considerations around computational overhead and resource allocation.

Overall, this chapter sets the stage for the contributions of this thesis which delves deeper into how these three domains interact and how they can be harnessed to push the boundaries of current machine learning capabilities.

3 - Asynchronous decentralized Bayesian optimization for large scale parallelism

---

As introduced in the previous chapter, sequential Bayesian optimization (BO, Section 2.2.4) is a promising approach for hyperparameter optimization of expensive learning workflows. We now look at deep neural networks (DNNs) learning workflows that can take minutes to hours to finish. In sequential BO (Algorithm 1), a computationally "cheap" surrogate model is employed to learn the relationship between hyperparameters and performance. The goal of this chapter is to efficiently parallelize such methods to speed up the computation. For this, we present an asynchronous-decentralized BO, wherein each worker runs a sequential BO and asynchronously communicates its results through shared storage. We scale our method without loss of computational efficiency with above 95% of worker's utilization to 1,920 parallel workers (full production queue of the Polaris supercomputer at the Argonne Leadership Computing Facility, Figure 3.1) and demonstrate improvement in model accuracy as well as faster convergence on the CANDLE benchmark from the Exascale computing project.

3.1 . Introduction to parallel hyperparameter optimization

Black-box optimization seeks to optimize a function based solely on input-output information. This problem is of particular interest in many scientific and engineering applications and is quite relevant to several machine-learning tasks. In the former case, the optimized black box is often the result of a complex simulation code, software, or workflow where we can get only the output for a given input configuration. In the latter, many learning algorithms are sensitive to hyperparameters, which cannot be inferred during the training process and often need to be adapted by the user based on the training data (Hutter et al., 2014a). Existing methods for solving black-box optimization can be grouped into model-based and model-free methods. In the former, a surrogate model for the black-box function is learned in an online fashion and used to speed up the search (Wild et al., 2015; Hutter et al., 2011; Bergstra et al., 2011; Hauschild and Pelikan, 2011). In the latter, the search navigates the search space directly without any explicit model (Olsson and Nelson, 1975; Poli et al., 2007; De Boer et al., 2005; Bäck and Schwefel, 1993; Rutenbar, 1989). These two groups of methods have their strengths and weaknesses. A key advantage of model-based over model-free methods is the sample efficiency w.r.t. the number of black box evaluations required by the search. Given the surrogate model, the search can quickly identify promising regions of the search space and find high-quality solutions faster (w.r.t. search iterations) than model-free methods can (Shahriari et al., 2016).

Bayesian optimization (BO) is a promising class of sequential optimization methods. It has been used in a wide range of black-box function optimization tasks (Shahriari et al., 2016; Bischl et al., 2017; Bartz-Beielstein, 2016). In BO, an incrementally updated surrogate model is used to learn the relationship between the inputs and outputs during the search. The surrogate model is then used to prune the search space and identify promising regions. BO navigates the search space by achieving a balance between exploration and exploitation to find high-performing configurations. While the exploration phase samples input configurations that can potentially improve the accuracy of the surrogate model, the exploitation phase samples input configurations that are predicted by the model to be high-performing.

With the transition of high-performance computing (HPC) systems from petascale to exascale (Heldens et al., 2020), massively parallel Bayesian optimizations that can take advantage of multiple computing units to perform simultaneous black-box evaluations are drawing attention to accelerate black-box optimization. These methods will be particularly beneficial for many HPC use cases, such as simulator calibration, software tuning, automated search of machine learning (ML) pipelines, neural network architecture, hyperparameter tuning (studied in this paper), and scientific simulation optimization. However, one of the main challenges is to transform commonly proposed sequential BO (Bergstra et al., 2011; Hutter et al., 2011; Bergstra et al., 2013; Klein et al., 2017) algorithms to be parallel while keeping a similar sample efficiency.

The most sample-efficient heuristic for parallel BO methods is a multipoint acquisition

strategy under a centralized architecture, where the manager performs BO and workers evaluate the black-box functions. Also, these methods often use a Gaussian process regression (GPR) as a surrogate model (Shahriari et al., 2016; Frazier, 2018). However, these two components respectively the centralized architecture and the GPR are two major bottlenecks for scaling BO in an HPC setting.

The problem we seek to solve is to **improve both solution quality and the speed of hyperparameter optimization (HPO)**, by optimizing **resource utilization of a large number of parallel computing workers**. To that end, we develop a parallel BO method based on a **decentralized architecture** (without a single manager). Each worker runs its own sequential BO and **communicates asynchronously** its hyperparameter evaluation results with other workers through a shared storage system. Additionally, each worker performs **asynchronous hyperparameter suggestion for the next evaluation**. Figure 3.1 shows the comparison between the centralized and our proposed approach for neural network hyperparameter tuning from the ECP Candle benchmark. Despite each evaluation taking several minutes, the centralized scheme suffers from poor utilization at scale. Our proposed decentralized method overcomes the limitation of a single-manager scheme and achieves high resource utilization. In Section 3.4, we will show that the higher utilization results in superior solution quality as well.

From the methodology perspective, our original contribution is to combine key algorithmic ingredients (decentralized, asynchronous surrogate model queries, qLCB acquisition function, periodic exponential decay) which benefit from increasing the number of workers to improve the final solution as well as reduce the time to this solution.

From the software perspective, our original contribution is first to provide an abstract interface to submit and gather black-box evaluations in combination with a shared stor-

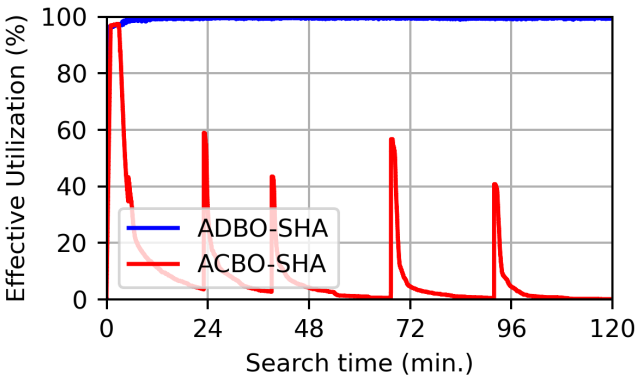


Figure 3.1: Utilization of computational resources between centralized (red) and decentralized (blue) Bayesian optimization equipped with Successive Halving (SHA) when using 1,920 workers (GPUs) (full Polaris HPC-system at the Argonne Leadership Computing Facility) to train neural networks configurations in parallel. The widely used single manager multiple worker utilization approach suffers from poor utilization. Our proposed decentralized approach maintains high utilization.

age service. Our job scheduler can leverage different backends for distributed computing (*e.g.*, threads, processes, MPI, Ray), similarly, our storage service can leverage different data services (*e.g.*, local memory, Redis, Ray Actors). We believe that this extensible architecture can open new research for HPC services tailored for large-scale hyperparameter optimization. Second, we provide a wrapper to launch seamlessly the method we propose with the Message Passing Interface (MPI) which could be valuable to run large-scale BO campaigns on many modern HPC systems.

Our **principal findings** support the advantage of using asynchronous decentralized BO:

1. **Stable resource utilization** when increasing the number of workers.
2. **Faster convergence and better solution** over the usual single manager and multiple workers when increasing the number of workers.
3. **Benefit from early discarding** strategies to speed up the procedure at a fixed computational scale without loss of computational performance.
4. **Parallel Bayesian Optimization at HPC scale** involving 1,920 workers (one NVIDIA A100 GPU per worker).

3.2 . Overview of parallel Bayesian optimization

Bayesian optimization is a well-established method to solve the global optimization problem of expensive and noisy black-box functions (Mockus et al., 1978). For a detailed overview see (Shahriari et al., 2016). In this chapter, we seek to solve the problem of hyperparameter optimization of a randomized learner (Problem 2.4). We do this from a black-box optimization perspective given by the following problem:

$$\theta^* = \underset{\theta \in \Theta}{\arg\min}\, f(\theta) \tag{3.1}$$

where $\theta \in \Theta$ is a hyperparameter vector, and $f(\theta) \in \mathbb{R}$ is a function that evaluates the learner for hyperparameters θ and returns a scalar value corresponding to the performance of the output predictor from this learner on a supervised learning task (Problem 2.2). We recall that Θ is a space of mixed variables (Section 2.1). Typically, the feasible hyperparameter set is defined by a set of constraints on θ or $f(\theta)$. This includes bound constraints that specify the minimum and maximum values for the parameters and linear and non-linear constraints that express the feasibility of the given hyperparameters through algebraic constraints. On the other hand, hidden constraints are unknown or hard to express and generally require an evaluation of the black-box function f to be uncovered. For example, it is hard to guess which hyperparameter configurations can result in an "out of memory" GPU error. The objective function $f(\theta)$ can be deterministic (the same values $f(\theta)$ for the same θ) or stochastic (different $f(\theta)$ values for the same θ). Generally, finding and prooving the true global optimal solution of Problem 2.4 is not possible (Larson et al., 2019; Bartz-Beielstein, 2016; Bischl et al., 2017), except for the simplest

cases. The presence of integer and categorical parameters, algebraic and hidden constraints, results in a highly irregular optimization landscape that makes the optimization process difficult. Several mathematical optimization algorithms take advantage of gradients that measure the change in the value of the objective function w.r.t. the change in the values of the parameters. This is not feasible in our setting as the black-box function cannot be differentiated.

3.2.1 . Review of surrogate models

BO uses a surrogate model assumed to be computationally cheaper than the black box to suggest the next points to evaluate. The choice of a good surrogate model plays a crucial role in the scalability and effectiveness of the BO search method. In most BO methods, a Gaussian process regression (GPR) is employed because of its sound uncertainty quantification capability (Shahriari et al., 2016; Frazier, 2018). Specifically, GPR implicitly adopts Bayesian modeling principles of estimating the posterior distribution of output from the given input-output pairs and provides the predictive mean and variance for the unevaluated input configurations. GPR also has the advantage of being differentiable. However, while GPR is superior for faster convergence when run sequentially on continuous optimization problems it remains one of the key bottlenecks for computational scalability in an HPC setting where thousands of input-output pairs (samples) can be computed in one "batch". The GPR model needs to be refitted with a rapidly growing set of samples (past and new), but it has a cubic complexity $O(n_{\text{sample}}^3)$ (Liu et al., 2019a) w.r.t. the number of samples n_{sample}. For a small n_{sample}, this is not an issue. At scale, however, the cubic time complexity for model fitting will slow or even stop the search's ability to generate new input configurations, thus increasing the idle time of the workers and eventually resulting in poor HPC resource utilization as well as fewer overall evaluations. For example, the popular Optuna (Akiba et al., 2019b; Optuna developers, 2018) hyperparameter optimization software advises to use BoTorch (Balandat et al., 2020) (a efficient implementation of BO based on GPR) for a range of 10 to 100 evaluations. Our largest parallel experimental setting, 1,920 parallel workers, corresponds to about x20 more evaluations in just one parallel iteration, therefore, making the use of a GPR surrogate impossible. Other surrogate models were proposed in the HPO literature such as deep neural networks (Snoek et al., 2015), Tree-Parzen estimation (TPE) (Bergstra et al., 2011), and random-forests regression (RFR) (Breiman, 2001; Geurts et al., 2006; Hutter et al., 2014b). We adopt RFR for its wide adoption thanks to its versatility with real, discrete, and categorical features as well as its robustness. RFR has a fitting time complexity of $O(n_{\text{tree}} \cdot n_{\text{feature}} \cdot n_{\text{sample}} \cdot \log(n_{\text{sample}}))$ with n_{tree} number of trees in the ensemble and n_{feature} number of features ($= d$, problem dimension) per sample, which is constant for each search setting. In addition to the log-linear time complexity, RFR provides simple and easy in-node parallelization opportunities, where each tree can be built independently of other trees in the ensemble. It is also important to note that RFR is not sensitive to the re-scaling of the input space (features) but is particularly sensitive to the re-scaling of the target space. We apply a $\log(.)$ transformation on the normalized (between $[\epsilon, 1]$ up to a sign depending on maximization/minimization) target space to improve the convergence of the BO to quantities of

interest.

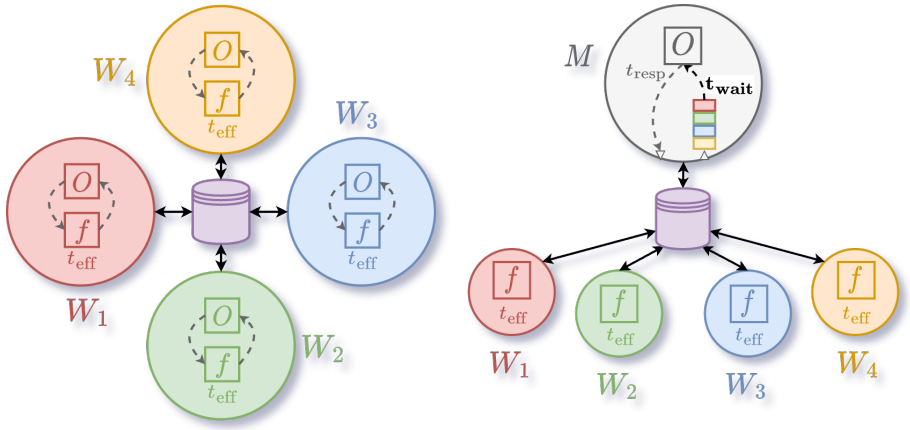
3.2.2 . Review of multipoint acquisition strategies

How the input point θ is selected for evaluation is another bottleneck for scaling the number of workers. The selection method comprises an acquisition function (Section 13) that measures how good a point is and a solver that seeks to optimize the acquisition function over θ (Section 13). Typically, when all the parameters are continuous (or if they afford such encoding/transformation), gradient-based optimizers are employed to select the next point. However, due to RFR not being differentiable such optimizers cannot be employed. While derivative-free (Section 2.2.3) solvers for these types of problems do exist, they are computationally expensive and cannot be employed in the fast iterative context required for scaling. Therefore, we use a point selection scheme with the lower confidence bound (LCB) (Shahriari et al., 2016) acquisition function on top of a random sampling from Θ. This scheme selects an input point θ for evaluation as follows. A large number of *unevaluated* configurations are sampled from the feasible hyperparameter space. The BO uses a dynamically updated surrogate model m to predict a point estimate (mean value) $\mu(\theta)$ and variance $\sigma(\theta)^2$ for each sampled configuration θ. The best sampled configurations has the smallest LCB score given by

$$a_{\text{LCB}}(\theta; \kappa) := \mu(\theta) - \kappa \cdot \sigma(\theta) \tag{3.2}$$

where $\kappa \geq 0$ is a parameter that controls the trade-off between exploration and exploitation. When κ is set to zero, the search performs only exploitation (greedy); when κ is set to a large value, the search performs stronger exploration. A balance between exploration and exploitation is achieved when κ is set to an appropriate value, classically $\kappa = 1.96$, which translates into a 95% confidence interval around the mean estimate when computing LCB.

In parallel BO, there are mainly two ways for querying $q > 1$ new suggestions in parallel. On the one hand, the centralized method tries to resolve a multipoint optimization problem such as the qEI criteria (Ginsbourger et al., 2010b; Shahriari et al., 2016). In this case, a manager runs the BO and generates configurations (González et al., 2016; Snoek et al., 2012), and the workers evaluate the configurations and return the results to the manager. The manager generates configurations in a batch synchronous or asynchronous way (Alvi et al., 2019). But, to generate these batches the multipoint optimization problem has to be solved and it becomes harder and more computationally expansive when q increases. Therefore, heuristics such as the constant-liar (CL) strategy (Ginsbourger et al., 2010b; Balaprakash et al., 2018b) have been used to approximate this criterion. Still, the CL scheme has a linear temporal complexity w.r.t. the number of workers that blocks from scaling when increasing the number of parallel workers. That is, for q new suggestions the surrogate model needs to be updated q times sequentially. In addition, when a configuration finishes (at any time) the previous batch can still be processed and therefore results in congestion in the manager's queue. In contrast to the centralized architectures, decentralized BO was recently introduced based on stochastic policies such as Thompson sampling and Boltzmann policy (Hernández-Lobato et al., 2017; Garcia-Barcos and



(a) Decentralized (b) Centralized

Figure 3.2: Decentralized (3.2a) and centralized (3.2b) search models with a shared storage. A circle represents a process with W for a worker and M for a manager, an arrow represents a communication, O represents the optimizer, and f represents the computation of the black-box function. t_{wait} is the time for which a worker waits before being processed by the optimizer. t_{resp} is the time taken by the optimizer to suggest a new configuration.

Martinez-Cantin, 2019). Stochastic policies allow bypassing communication in the decentralized architecture, which enables asynchronous iterations. However, the asynchronous case was not studied in these works.

3.3 . An asynchronous and decentralized method to scale parallel Bayesian optimization

In this section, we introduce our novel approach to BO in the context of HPC exploiting a decentralized architecture with asynchronous communication. Multiple compute units can be used on an HPC platform, where each function evaluation requires a fraction of this platform. Let n_{worker} be the number of available workers, where a worker represents a unit of computational resource available to evaluate the black-box function (*e.g.*, CPU, GPU, a fraction of a compute node, whole node, or a group of nodes). Let T_{wall} be the total wall-clock time for which these resources are available (*i.e.*, job duration). Then the overall available compute time $T_{\text{avail}} = n_{\text{worker}} \cdot T_{\text{wall}}$ upper bounds the total time spent in black-box function evaluations $T_{\text{eff}} = \sum_{t_{\text{eff}} \in \mathcal{T}} t_{\text{eff}}$ used to perform the job, where $\mathcal{T}$ is the set of duration t_{eff} for all evaluated black-box functions. We define "effective utilization" as $U_{\text{eff}} = T_{\text{eff}}/T_{\text{avail}}$. For the problem of "parallel black-box optimization", we seek to maximize the objective function f, as well as maximize the effective utilization U_{eff}. We maximize utilization by minimizing the computational overhead of the search algorithm and assume that the quality of the optimization will be a byproduct of better utilization. In other words, in addition to the objective function maximization, the optimization method

should effectively use parallel resources by maximizing their usage mainly with black-box evaluations.

BO is a promising approach for tackling the class of black-box optimization problems described in Section 3.2. BO tries to leverage accumulated knowledge of f throughout the search by modeling it as a probability distribution $\mathrm{P}\left(C|\Theta\right)$, which represents the relationship between θ, the input, and c, the output cost. Typically, BO methods rely on dynamically updating a surrogate model that estimates $\mathrm{p}\left(c|\theta\right)$. Often, this distribution is assumed to follow a normal distribution (Section 13). Therefore, the surrogate model estimates both $\mu(\theta)$ the mean estimate of c and $\sigma(\theta)^2$ the variance. The latter is leveraged to assess (1) the distribution variance of $\mathrm{P}\left(C|\Theta\right)$ (*i.e.*, the noise of the black box function) and (2) how uncertain the surrogate model is in predicting $\mu(\theta)$ (Shahriari et al., 2016). The surrogate model is cheap for prediction and can be used to prune the search space and identify promising regions, where the surrogate model is then iteratively refined by selecting new inputs that are suggested by the model to be high-performing (exploitation) or that can potentially improve the quality of the surrogate model (exploration). BO navigates the search space by achieving a balance between exploration and exploitation to find high-performing input configurations.

3.3.1 . Asynchronous decentralized Bayesian optimization

Figure 3.2a shows a high-level sketch of our proposed asynchronous decentralized Bayesian optimization (ADBO) method. The key feature of our method is that each worker executes a sequential BO search (recall Algorithm 1); performs only one black-box evaluation that avoids congestion occurring in a centralized setting (see Figure 3.2b); and communicates the results with all other workers in an asynchronous manner through shared storage. The BO of each worker differs from that of the other workers w.r.t. the value κ_0 used in the LCB acquisition function. Each BO starts by sampling the value κ_0 from an exponential distribution $\exp(\frac{1}{\kappa})$, where κ is the user-defined parameter. The BO that takes smaller κ_0 values will perform exploitation and sample points near the best found so far in observations. On the other hand, the BO that receives large κ_0 values will perform exploration to reduce the predictive uncertainty of the RFR model. Consequently, on average multiple BO searches will seek to achieve a good trade-off between exploration and exploitation; however, there will be several BO searches that perform stronger exploitation or exploration. This effect will increase as we scale to a large number of workers and can benefit the overall search. Our approach is inspired by the qLCB acquisition function (Jones, 2001; Hutter et al., 2012) where different κ values are sampled from the exponential distribution for the LCB acquisition function and different points are selected based on these values to find the balance between exploration and exploitation. The main reason for adopting qLCB is computational simplicity. Compared to other multipoint generation strategies, such as the constant liar (Ginsbourger et al., 2010b) (denoted by CL) that exist mainly in the centralized (single-manager/multiple-worker) methods and has an overhead increasing linearly with the number of workers, there is only an overhead constant in the number of workers when using qLCB.

Algorithm 2 shows the high-level pseudocode of our proposed ADBO method. As it can

be seen it is a direct decentralized parallel extension of the sequential BO (Algorithm 1). Each worker runs a sequential BO and communicates their observations with each other. The search proceeds by initializing the surrogate model (line 2) and the initial κ_0 with a random sample from an exponential distribution with mean parameter $1/\kappa$ (line 3).

Then, we enter the sequential optimization loop (line 5). At the beginning of each loop, the current κ_t is updated following the exponential decay schedule (line 6). The selection of the next hyperparameter is done based on κ_t (lines 8-15) and it is evaluated accordingly. To handle failed evaluations, such as tested hyperparameter configurations that returned a memory error, we replace the missing objective values with the maximum observed cost (line 11). Completed evaluations are exchanged between workers (lines 18-19). In our case, we implemented this communication through a fast in-memory database service represented in purple in Figure 3.2a. Local data and remote data are concatenated (lines 21-22). The recommended hyperparameters are updated (line 23). Finally, we increment the local iteration counter (line 24).

Algorithm 2: Asynchronous Decentralized Bayesian Optimization (Worker Process)

Inputs : `thetaSpace`: a configuration space,
`nInitial`: the number of initial hyperparameter configurations,
`f`: a function that returns the cost of the learning workflow,
κ: the parameter of qLCB,
T: the period of the exponential decay,
λ: the decay rate of the exponential decay
Output: `thetaStar` the recommended hyperparameter configuration.

1 `thetaArray`, `costArray` ← New empty arrays of hyperparameter configurations and costs ;
2 `model` ← New RFR surrogate model ;
3 $\kappa_0 \leftarrow$ Sample from exponential distribution with parameter $1/\kappa$;
4 $t \leftarrow 0$;
5 **while** *compute time is not exhausted* **do**
`/* Apply exponential decay */`
6 $\kappa_t \leftarrow \kappa_0 \cdot \exp(-\lambda \cdot ((t - \texttt{nInitial}) \bmod T))$;
7
8 **if** *Length of* $\texttt{thetaArray} < \texttt{nInitial}$ **then**
9 `theta` ← Sample hyperparameter configuration from `thetaSpace` ;
10 **else**
11 `costArrayAux` ← Returns new array where failures in `costArray` are replaced with $\max(\texttt{costArray})$;
12 Update `model` with `thetaArray`, `costArrayAux` ;
13 `theta` ← Returns `theta` in `thetaSpace` that minimizes $a_{\text{LCB}}(\theta; \kappa_t)$ for current `model` through random sampling ;
14 **end**
15 `cost` ← Returns the cost of learning workflow `f(theta)` ;
16
`/* Exchange observations */`
17 `thetaArrayNew`, `costArrayNew` ← Collects new observations from other workers ;
18 Share new observation `theta`, `cost` with other workers ;
19
20 `thetaArray`, `costArray` ← Concatenate `thetaArray` with `thetaArrayNew` and `costArray` with `costArrayNew`;
21 `thetaArray`, `costArray` ← Concatenate `thetaArray` with [`theta`] and `costArray` with [`cost`];
22 `thetaStar` ← Update recommendation ;
23 $t \leftarrow t + 1$;
24 **end**

3.3.2 . Extremely randomized trees surrogate model

Although RFR provides computational advantages, its uncertainty quantification capabilities are not well-known or documented in the literature. The most widely used RFR implementation is from the Scikit-Learn package (Pedregosa et al., 2011). Our analysis of uncertainty quantification with the default implementation of this package showed that the predictive variance is not as good as that of GPR (Figure 3.3a). The primary reason was the best-split strategy adopted in the usual random forest algorithm to minimize the variance of the estimator. Although this results in better predictive accuracy, the predictive variance is not informative in the context of Bayesian optimization because it remains constant in unexplored areas. We tested the random-split strategy for tree splitting, as suggested in (Hutter et al., 2014b), and found that the uncertainty estimates are improved and are comparable to those with GPR in the interpolation area while remaining constant in extrapolation areas (Figure 3.3b). It is good to note that doing so makes the Random-Forest close to the Extremely Randomized Trees model (Section 3.2.1) also available in Scikit-Learn. We believe that this confusion between Random-Forest (with best-split from Breiman (2001)) and Extremely randomized forest (with random-split from Geurts et al. (2006)) might be one of the reasons that Random-Forests (RFR, general class of models using ensemble of randomized trees), despite their computational advantages, were not thoroughly experimented with for uncertainty quantification. The uncertainty of the RFR is computed by applying the law of total variance (Theorem 3.1).

Theorem 3.1 (Law of Total Variance)**.** *If X and Y are random variables on the same probability space, and the variance of Y is finite then:*

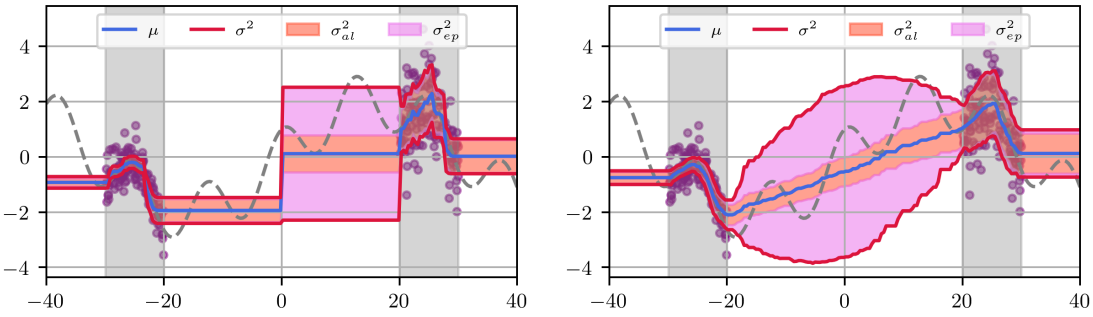
$$\mathbb{V}[Y] = \underbrace{\mathbb{E}_X[\mathbb{V}_Y[Y|X]]}_{\textit{aleatoric}} + \underbrace{\mathbb{V}_X[\mathbb{E}_Y[Y|X]]}_{\textit{epistemic}}$$

where the left term (aleatoric) is the "unexplained"-variance and the right term (epistemic) is the "explained"-variance.

The conditional mean $\mathbb{E}_C[C|\Theta = \theta, \texttt{tree}]$ and variance $\mathbb{V}_C[C|\Theta = \theta, \texttt{tree}]$ of the cost C given hyperparameters θ are estimated through each tree and respectively denoted as $\mu_{\texttt{tree}}(\theta)$ and $\sigma_{\texttt{tree}}(\theta)^2$. Therefore we have:

$$\sigma(\theta)^2 = \mathbb{E}_{\texttt{tree}}\left[\sigma_{\texttt{tree}}(\theta)^2\right] + \mathbb{V}_{\texttt{tree}}\left[\mu_{\texttt{tree}}(\theta)\right] \tag{3.3}$$

where $\mathbb{E}_{\texttt{tree}}[.]$ and $\mathbb{V}_{\texttt{tree}}[.]$ are respectively the empirical mean and variance. We denote by $\texttt{tree}$ the set of random effects that impact the construction of each tree (*e.g.*, bootstrapping, random-split, random feature selection). In Figure 3.3, we represented in purple σ^2_{ep} the epitemic uncertainty and in orange σ^2_{al} the aleatoric uncertainty. Later in Chapter 6, we will expand on these two types of uncertainties and show how they can be used at the level of the learning workflow instead of the optimization level such as done in the current chapter.



(a) Random Forest (b) Extremely Randomized Trees

Figure 3.3: Comparing uncertainty quantification of Random Forests.

We also compared the speed of RFR with improved uncertainty (based on Scikit-Learn) against the Python package `pyrfr`[1] used by (Hutter et al., 2014b). We observed that our implementation is orders of magnitude faster. For a test case with a single continuous input variable and a single continuous output target with 10,000 samples (2/3 training and 1/3 test), ours takes 0.12 seconds while the `pyrfr` implementation takes 24 seconds. The implementation based on Scikit-Learn can also benefit from multi-processing. Our new implementation of RFR with its improved uncertainty estimate module is made available as part of DeepHyper (Balaprakash et al., 2018a), open-source software for AutoML research on HPC.

3.3.3 . Preventing stagnation with periodic exponential decay

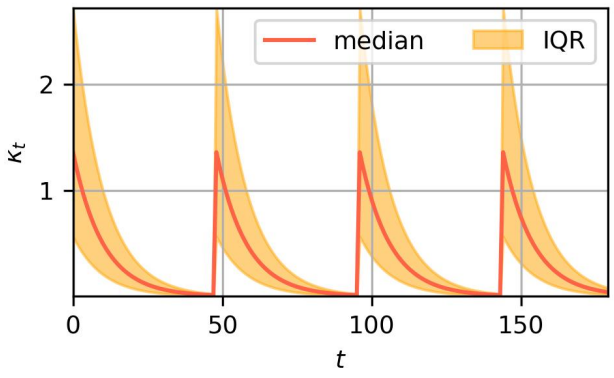


Figure 3.4: Illustration of the distribution of periodic exponential decay.

We propose a heuristic to dynamically manage the trade-off between exploration and exploitation of the qLCB in ADBO. For this, we rely on the idea of exponential decay for

[1] https://github.com/automl/random_forest_run

the κ parameter of the qLCB acquisition function, given by:

$$\kappa_{\lambda,T}(t;\kappa_0) = \kappa_0 \cdot \exp(-\lambda \cdot (t \bmod T) \tag{3.4}$$

where κ_0 is the initial exploration-exploitation trade-off, t is the number of local evaluations performed in the process, T is the period of the scheduler and λ is the decay rate of the scheduler. From this definition, in the case of ADBO, the κ_0 is a random variable sampled from an exponential distribution $\exp(1/\kappa)$ where κ is the mean of the distribution. Therefore we have the following average behavior for κ across processes in the synchronized case:

$$\mathbb{E}_{\kappa_0 \sim \mathsf{Exp}(1/\kappa)}\left[\kappa_{\lambda,T}(t;\kappa_0)\right] = \kappa \cdot \exp(-\lambda \cdot (t \bmod T)) \tag{3.5}$$

We provide an illustration of the distribution of $\kappa_{\lambda,T}(t;\kappa_0)$ with respect to t in Figure 3.4.

With such a scheduler, the different processes are periodically converging to the "exploitation" regime (small κ_t values) while the original "exploration" is regularly recovered. This heuristic avoids "over-exploring" (*i.e.*, behavior similar to random search) when the number of parallel workers is increased.

3.3.4 . Early discarding with asynchronous successive halving

We propose to combine our Bayesian Optimizer with an early discarding strategy. Early discarding strategies are methods used to discard early a hyperparameter configuration if it is not likely to improve over the best objective observed so far. In this work, we use the asynchronous successive halving (SHA) algorithm (Li et al., 2020). The SHA algorithm compares the performance of a new configuration with past evaluations. This comparison is done at different "rounds" allocated according to a geometric progression (based on training epochs such as 1, 3, 9, 27...). An algorithm is stopped if it is not among the top-$\frac{1}{\rho} \times 100\%$ at the current round where ρ is called the reduction factor. Later, in Chapter 5, we instigate more about early discarding strategies.

3.4 . Experimental results

We conduct an empirical study to demonstrate the benefits of using a decentralized architecture. First, we analyze the effectiveness of the two approaches by looking at the number of idle resources as well as the number of completed neural network training for a fixed computational budget (*i.e.*, number of workers and time of execution). Second, we analyze the gain in the final objective and the speed of the HPO.

Platform and main libraries – All the experiments are performed on the Combo benchmark from the Exascale-Computing Project. They are conducted on the Polaris supercomputer at the Argonne Leadership Computing Facility (ALCF). Polaris is an HPE Apollo Gen10+ platform that comprises 560 nodes, each equipped with a 32-core AMD EPYC "Milan" processor, 4 Nvidia A100 GPUs, and 512 GB of DDR4 memory. The compute nodes are interconnected by a Slingshot network. In our experiments, each worker is attributed 1 GPU. The algorithm is implemented in Python 3.8.13 where the main packages

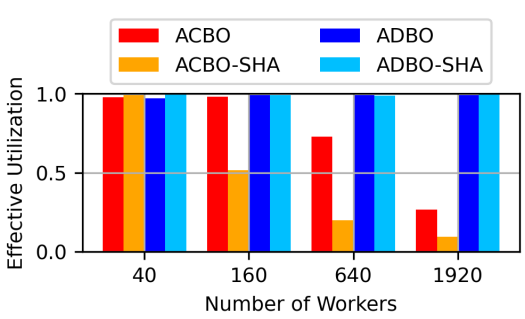
used are deephyper 0.5.0, mpi4py 3.1.3, scikit-learn 1.1.2, Redis 7.0.5. Neural networks are implemented in Tensorflow 2.10. The number of workers is increased from 40 (10 nodes), 160 (40 nodes), 640 (160 nodes) to 1920 (480 nodes – the full system in the production queue).

Benchmark – The Combo benchmark dataset (Xia et al., 2018) is composed of 165668 training data points (60%), 55,222 (20%) validation data points and 55,222 (20%) test data points respectively. Each data point has three types of input features: 942 (RNA-Sequence), 3,839 (drug-1 descriptors), and 3,893 (drug-2 descriptors) respectively. The data set size is about 4.2 GB. It is a regression problem to predict the growth percentage of cancer cells given cell line molecular features and the descriptors of two drugs. The networks are trained for a maximum budget of 50 epochs and 30 minutes. The negative validation R^2 coefficient at the last trained epoch is used as the objective for hyperparameter search.

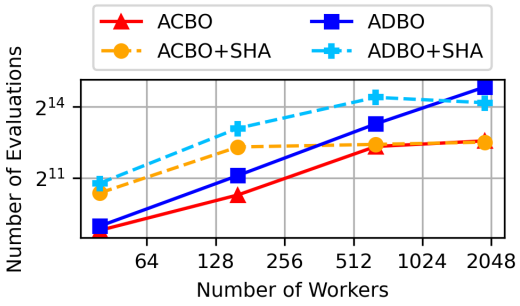
The baseline model is composed of 3 inputs each processed by a sub-network of three fully connected layers. Then, the outputs of these sub-models are concatenated and input in another sub-network of 3 layers before the final output. All the fully connected layers have 1000 neurons and ReLU activation. It reaches a validation and test R^2 of 0.87 after 100 epochs. The number of neurons and the activation function of each layer are exposed for the hyperparameter search. The search space is defined as follows: the number of neurons in $[10, 1024]$ with a log-uniform prior; activation function in [elu, gelu, hard sigmoid, linear, relu, selu, sigmoid, softplus, softsign, swish, tanh]; optimizer in [sgd, rmsprop, adagrad, adadelta, adam]; global dropout-rate in $[0, 0.5]$; batch size in $[8, 512]$ with a log-uniform prior; and learning rate in $[10^{-5}, 10^{-2}]$ with a log-uniform prior. A learning rate warmup strategy is activated based on a boolean variable. Accordingly, the base learning rate of this warmup strategy is searched in $[10^{-5}, 10^{-2}]$ with a log-uniform prior. Residual connections are created based on a boolean variable. A learning rate scheduler is activated based on a Boolean variable. The reduction factor of this scheduler is searched in $[0.1, 1.0]$, and its patience in $[5, 20]$. An early-stopping strategy is activated based on a Boolean variable. The patience of this strategy is searched in $[5, 20]$. Then, batch normalization is also activated based on the Boolean variable. The loss is searched among [mse, mae, logcosh, mape, msle, huber]. The data preprocessing is searched among [std, minmax, maxabs]. This corresponds to 22 hyperparameters. All experiments are performed with the same initial random state 42.

Metrics – During HPO the objective minimized is the negative coefficient of determination $-R^2$ on a hold-out validation dataset. Then, for our analysis, we use the regret defined as $1 - R^2$ where 0 is the lower bound of the objective. The model selection is always based on the validation scores but the test scores are presented to consider the problem of generalization in ML. To compare the speed of convergence between experiments as well as the quality of the solution we compute the Area Under the Regret Curve (AURC) defined in (Liu et al., 2021) but without re-scaling of the time. The smaller the AURC the best is the any-time performance. Both the regret and the AURC are common metrics used in AutoML (Liu et al., 2021; Eggensperger et al., 2021a).

3.4.1 . Efficient utilization of computational resources



(a) Effective utilization of workers.



(b) Number of evaluated hyperparameter configurations.

Figure 3.5: Comparison of (3.5a) effective utilization and (3.5b) number of completed hyperparameter evaluations of centralized/decentralized BO without/with Successive Halving (SHA) for an increasing number of parallel workers (1 GPU per worker). Overall, the decentralized approach is better with higher utilization and the number of evaluated configurations. Conversely, the utilization and number of evaluations quickly drop for the centralized approach.

We study the effectiveness of the centralized and decentralized architectures when the number of workers is increased. The general idea of looking at resource utilization is to detect overheads in parallel algorithms which when excessive can be counterproductive and worsen the results even though resources were increased. For this, we compute the effective utilization which we defined as the ratio between the cumulative time spent in training neural networks and the allocated total time ($=$ Number of Workers $\times$ Execution Time). The resource utilization w.r.t. the number of parallel workers is presented in Figure 3.5a. The centralized architecture has a utilization similar ($> 90\%$) to the decentralized at a small scale (40 workers). However, when the number of workers increases utilization drops quickly and finishes below 25% meaning that more than 75% of allocated resources are not being used. In addition, using an early discarding strategy makes the computational efficiency even worse as it can be observed that from 160 workers Cent. SHA (orange) has about half the utilization of Centralized (red). This behavior is normal as early discarding shortens the duration of evaluations by stopping early non-promising networks which increases the number of queries received by the manager, resulting in more frequent overhead.

However, looking at the utilization is not sufficient as it could be kept artificially high just by submitting hyperparameter configurations which take longer to be completed and therefore bypass the problem of querying frequently the BO agent. Therefore, we propose to also look at the number of completed evaluations, which is shown in Figure 3.5b. The number of completed evaluations is significantly higher for decentralized executions than for centralized ones. For 1,920 workers, the decentralized completed 29,222 evaluations while the centralized completed only 6,055 evaluations. Similarly Dist. SHA completed 18,431 while Cent. SHA completed only 5,832 evaluations. The number of evalua-

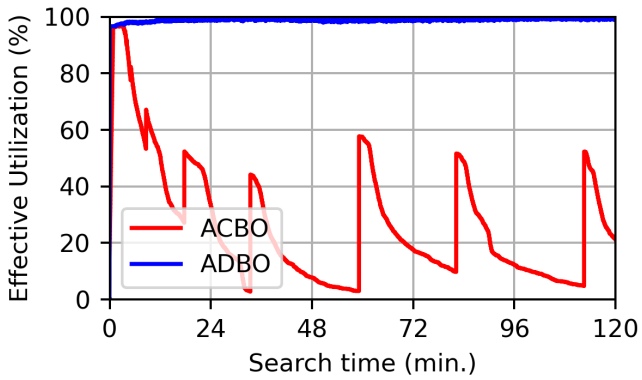


Figure 3.6: Comparing the evolution of worker utilization for the centralized and decentralized BO with a total of 1,920 parallel workers. Overall, the decentralized approach has high utilization while the centralized approach drops early on without ever recovering.

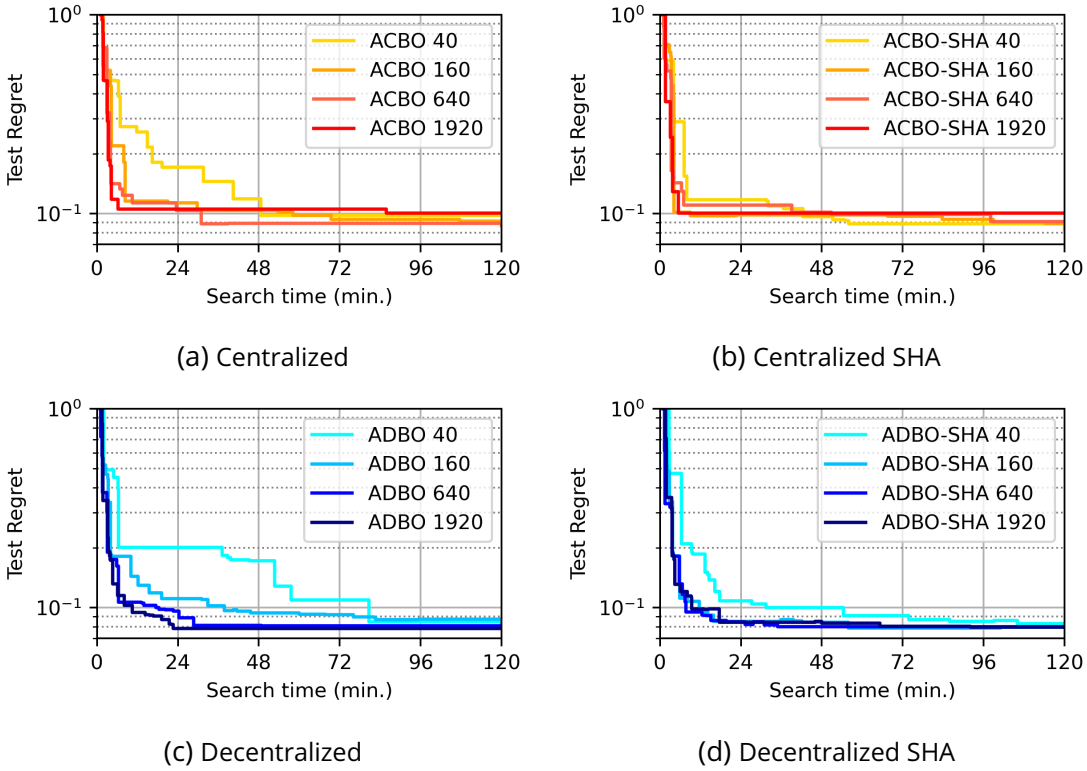
tions increases linearly for the decentralized, unlike the Centralized and Cent. SHA which plateau. The behavior of Dist. SHA with 1,920 workers can be explained through the BO agent that suggests in this case hyperparameter configurations that are longer to train. By looking at the number of evaluations we showed that under frequent queries the search can maintain high effective utilization.

Finally, to explain the drop in utilization of centralized executions we take a closer look at the profile of the utilization during execution. The profiles are presented in Figure 3.6 and Figure 3.1 respectively for the black-box and gray box settings. It can be observed that the Centralized starts to lag after only 10 minutes of execution (when the first results are received) and displays some oscillations later on which are the results of completed batches of received queries. However, these queries come in too quickly to be processed and the manager is overloaded which results in congestion and a drop in utilization that can never be recovered. The same type of profile can be observed in the Cent. SHA case as well as in previous literature (Balaprakash et al., 2018b).

3.4.2 . Better and faster hyperparameter optimization

After analyzing the computational performance of ADBO compared to ACBO we now evaluate the gain in "search quality" when the number of workers is increased. Increasing the quantity of computational resources should speed up the HPO process as well as improve the final results (*i.e.*, the returned solution has better predictive capabilities).

The first thing we analyze is the "search trajectory" which corresponds to the evolution of the test regret w.r.t. the execution time such as presented in Figure 3.7. Intuitively, the search trajectory can help judge the quality of an HPO algorithm through at least two aspects: first the solution after convergence, and second the time it takes to reach this solution. We start by providing a qualitative analysis of these figures which is later supported by quantitative metrics. It is clear from the centralized setting in Figure 3.7a and 3.7b that increasing the number of workers has a negative impact on the quality of the



(a) Centralized

(b) Centralized SHA

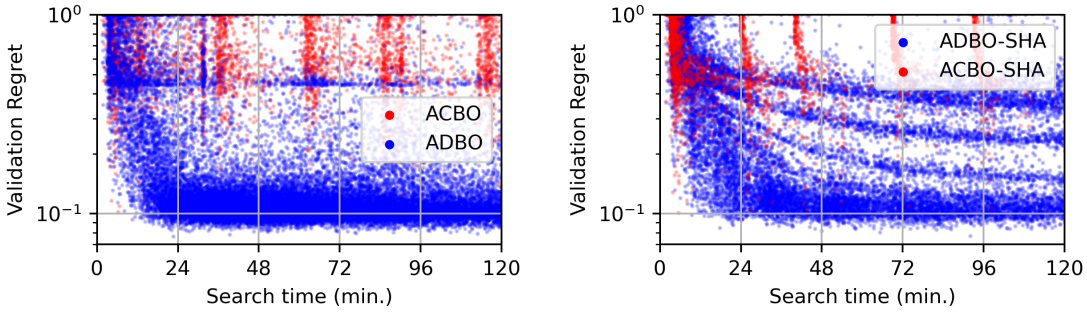
(c) Decentralized

(d) Decentralized SHA

Figure 3.7: Comparing the "search trajectory" (*i.e.*, the evolution of the test regret w.r.t. the execution time) for the centralized (top, 3.7a and 3.7b) and decentralized (bottom, 3.7c and 3.7d) implementations with and without successive halving (SHA). Overall, the decentralized scales consistently where each search trajectory with a larger number of workers dominates or performs similarly to smaller scales. A similar performance in regret can be explained by the benchmark reaching saturation (*i.e.*, no better objective can be found). On the contrary, the centralized approach becomes inefficient at large scales and performs worse than at smaller scales.

search. Indeed, when using 1,920 workers even though the early iterations are efficient, which confirms the "sample efficiency" of the constant-liar strategy, the trajectory reaches a plateau early on which is surpassed by executions with fewer workers. However, in the case of the decentralized setting in Figure 3.7c and Figure 3.7d the trajectories corresponding to a larger number of workers are dominating trajectories with fewer workers. It means that it is faster to get to the same solution and the final solution is better.

Now, the search trajectory is not all we want to observe because it does not display the "strength" of convergence but only the quality of iterative improvements. Therefore, the online observations of the execution at the largest scales are presented through scatter plots in Figure 3.8a and 3.8b. Through these figures, it is clear that the decentralized variants converged to an area of the search space consistently suggesting better hyperparameter configurations (the baseline has a performance of 0.125 in test regret). On the contrary, the centralized variants seem to rarely suggest good configurations. The



(a) Black-Box Bayesian Optimization.

(b) Gray-Box Bayesian Optimization.

Figure 3.8: Comparing the observations of centralized (red) and decentralized (blue) approaches for both (a) black-box and (b) gray-box (with Successive Halving – SHA) settings at a scale of 1,920 parallel workers.

capability of a search to discover different hyperparameter configurations and therefore neural network architectures can be used to assess the uncertainty in model-choice (aka epistemic uncertainty) and reduce the variance of estimation through "ensembling" such as proposed in (Wenzel et al., 2020; Égelé et al., 2022) and later presented in Chapter 6. The behavior of SHA can also be observed through the stratification of the y-axis in Figure 3.8b (*i.e.*, early termination of low-performing configurations).

Finally, from a quantitative point of view, we summarize the AURC and minimum test regret for all the methods in Table 3.1 over the different scales of parallel workers. The performance of random search (without early discarding) is presented to complement our previous sanity checks. Also, random search scales without issues as communication is not required and is a good competitor when increasing the number of workers. At a

	40 Workers		**160 Workers**		**640 Workers**		**1920 Workers**	
Method	AURC	Min. Regret	AURC	Min. Regret	AURC	Min. Regret	AURC	Min. Regret
Random	0.234	0.185	0.156	0.116	0.128	0.104	0.12	0.096
ACBO	0.154	0.098	0.126	0.092	0.116	0.089	0.122	0.100
ADBO	0.162	0.085	0.120	0.087	0.102	0.081	**0.099**	**0.079**
ACBO-SHA	**0.125**	0.089	0.118	0.089	0.119	0.091	0.117	0.1
ADBO-SHA	0.132	**0.083**	**0.105**	**0.080**	**0.099**	**0.08**	0.103	0.08

Table 3.1: Summarizing the results across all the experiments for the Combo Benchmark. The centralized is colored in red and the decentralized is colored in blue. The *"AURC"* represents the Area Under the (test) Regret Curve w.r.t. the time. The *"Min. Regret"* is the value of the test regret corresponding to the best validation objective observed by the search. Smaller values of AURC and Min. Regret are better. The best scores across methods are in bold font. Overall, **at the smallest scale the centralized is the most efficient (smallest AURC)** which confirms the sample efficiency of the constant-liar strategy. However, **when increasing the scale the decentralized is better** with faster convergence and smaller regret.

small scale (40 workers) the centralized has a smaller AURC with a regret very close to the decentralized. This confirms that at a small workers' scale, the centralized approach is efficient and legitimate. Then, when increasing the number of workers (from 160 to 1,920) the decentralized consistently outperforms the centralized settings. Also interesting, when we reach the largest scale of workers we enter a regime where "compute" is not a bottleneck for HPO anymore and we can observe that ADBO (maximum fidelity) is outperforming ADBO+SHA (early discarding).

3.5 . Conclusion

The issue we address is the task of optimizing resource utilization for large-scale HPO. We consider this problem to be of importance based on the last generation of released HPC systems that are equipped with at least 1,000 accelerator chips (*e.g.*, GPUs) and the popularity of ML in science. In this study, we focus on a well-adopted BO procedure, based on a Random-Forest surrogate model (Hutter et al., 2011), which already demonstrated its efficiency in solving the HPO problem both sequentially and at small scales of parallelism. Then, we focus on the asynchronous querying of the BO agent as the training time of neural networks can vary depending on their hyperparameter configurations (*e.g.*, learning rate, batch size, number of weights) and it was already shown to improve worker utilization (Li et al., 2020). We study the role of centralized and decentralized architectures of parallel BO. Indeed, centralized approaches are known to be more sample-efficient. However, the agent quickly becomes overloaded when the number of workers increases Therefore we developed a decentralized BO procedure with a custom acquisition function heuristic to maintain efficiency. We demonstrated at large scales (up to 1,920 workers), both in the case of black-box and gray-box optimization, that this approach improves significantly the utilization of computational resources compared to the centralized setting. We presented the profiles of worker utilization to explain our results. Finally, we summarized our results over different scales of parallel workers with scalar metrics (the "Area Under the Regret Curve" and the "Minimum Regret") to demonstrate the advantage both in the speed of convergence and quality of the solution when using the decentralized BO.

During our investigations, we considered for comparison the most popular available frameworks for distributed HPO such as RayTune (Liaw et al., 2018) and Optuna (Akiba et al., 2019b). Both had issues when scaling to the full Polaris system. For RayTune, the issue was to initialize the Ray cluster which resulted in a significant overhead and with many workers failing to connect to the main server. For Optuna configured with PostgreSQL and TPE sampler, when running properly at smaller scales (up to 640 workers) the best objective was similar to our proposed ADBO but resource utilization was always significantly lower and impacted by the scale. In this case, the limiting factor triggering failures was the number of parallel open connections to the database (larger than the number of workers).

One limitation of this study is the metric of worker utilization, measuring the time

spent computing the black-box function which does not accurately reflect all the bottlenecks of the BO procedure when increasing workers such as the overheads associated with parallel training of neural networks. These overheads, such as parallel I/O (*e.g.*, reading data and checkpoint weights), can create latency that is not captured in the worker utilization metric. Another important limitation of the paper is that the speed-up achieved by scaling workers in HPO is problem-dependent and may vary based on factors such as the dataset, hyperparameter search space, and ML pipeline (*e.g.*, maximum number of training epochs). Furthermore, the lack of repeated experiments is also to be noted with the known variability of training neural networks making the results in objective quality inconclusive. Lastly, the paper does not evaluate Gaussian processes regression due to their quick limitations w.r.t. the number of observations. Overall, this paper provides insights into the challenges associated with scaling the number of workers for HPO using BO procedures.

We are making our work available to the community as part of existing open-source software[2]. In our future work, we aim to transfer the discovered benefits of this study to other applications such as simulation calibration, software, and workflow tuning. Other areas of improvement are 1) the use of accelerators for surrogate model updates, 2) low overhead optimization of the acquisition function instead of random sampling, 3) domain decomposition of the search space, 4) optimization of parallel I/O due to repeated neural network training, and 5) multi-objective optimization.

3.6 . Acknowledgment

This research was conducted in collaboration with Isabelle Guyon, Venkatram Vishwanath, and Prasanna Balaprakash. This material is based upon work supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility. This material is based upon work supported by ANR Chair of Artificial Intelligence HUMANIA ANR-19-CHIA-0022 and TAILOR EU Horizon 2020 grant 952215.

[2] deephyper.readthedocs.io

4 - Multi-objective hyperparameter optimization with uniform normalization and bounded objectives

---

In the previous chapter, we presented a scalable decentralized Bayesian optimization for large-scale parallel hyperparameter optimization. This method could optimize only a single objective such as accuracy. While accuracy is a commonly used performance objective, in many settings, it is not sufficient. Optimizing the learning workflows for multiple objectives such as accuracy, confidence, fairness, calibration, privacy, latency, and memory consumption is becoming crucial. To that end, in this chapter, we adapt our decentralized Bayesian optimization to the multi-objective setting. Differences in objective scales, the failures, and the presence of outlier values in objectives make the problem even harder than for a single objective. We propose to address these problems through uniform objective normalization and randomized weights in scalarization. We increase the efficiency further by imposing constraints on the objective to avoid exploring unnecessary configurations (e.g., insufficient accuracy). Finally, we experiment with our multi-objective strategy jointly with our decentralized BO to demonstrate its scalability.

4.1 . Introduction to multi-objective hyperparameter optimization of learning workflows

Even though the field of HPO has made significant advances in the context of single-objective optimization it seems now insufficient to consider only one objective as the set of competing objectives (metrics) to improve has grown. In recent years, it has become necessary for many applications to consider additional metrics, such as privacy (Dwork, 2008), bias and social fairness (Mehrabi et al., 2021), calibration (Song et al., 2021), predictive uncertainty (Begoli et al., 2019; Gawlikowski et al., 2023), explainability (Burkart and Huber, 2021; Tjoa and Guan, 2020), adversarial robustness (Muhammad and Bae, 2022), temporal and memory complexity (Tan et al., 2019). Equally important, recent progress in ML was possible thanks to an increase in parallel computation (Jordan and Mitchell, 2015). As the multi-objective problem is harder than single-objective, a promising approach to improve consists in scaling the search to leverage multiple compute units (such as GPUs) which we study in this work.

After reviewing the current multi-objective hyperparameter optimization (MOHPO) literature we identify a few gaps. First, we notice that the strongest contenders such as NSGA-II, a variant of genetic algorithms, lack iteration efficiency (*i.e.*, improvement per completed black-box evaluations). This is important for expensive black-box such as in HPO. Second, we notice that most methods do not take into account practical considerations such as exploring only interesting trade-offs. For example, let us imagine that we wish to develop a model that is accurate and fast to query. It can be that the accuracy is so low that such a model (even if faster) would not be a good candidate for any other metric (here the speed of inference). Third, we notice that surrogate-based methods often do not provide solution diversity (w.r.t. hypervolume defined in a later section) and require specific adaptation and customization (*e.g.*, choice of scalarization function, trade-off weights, normalization of objectives, tuning of a surrogate). Finally, we notice that some surrogate-based methods do not benefit from scaling to more parallel resources as they do not gain significantly in performance or can not keep up with the demand due to their temporal complexity (*e.g.*, methods using Gaussian-process models).

We focus on developing a method for MOHPO that addresses the presented limitations: (a) improving iteration efficiency, (b) only exploring objectives trade-offs of interest, and (c) using parallelism to improve the effectiveness of the HPO. To that end, we develop a parallel search approach called decentralized multi-objective Bayesian optimization (D-MoBO) that combines an extremely randomized trees surrogate model (a type of Random-Forests), a randomized scalarization, a quantile transformation of objectives, and a penalty function. The parallelism results from the fact that we extend Algorithm 2 from the previous Chapter 3. The **methodological contributions** are:

1. A **quantile uniform normalization of objectives** gives more importance to the approximated Pareto-Front in the hypervolume indicator and is also robust to outliers.

2. A **penalty** to avoid "out of interesting range" objective trade-offs.

3. A **parallel implementation** boosts optimization performance.

4.2 . Overview of general multi-objective optimization

In a generic multi-objective optimization (MOO) problem, the cost function is a vector-valued $f : \Theta \to \mathcal{C}$, $\mathcal{C} = \mathbb{R}^{d_c}$, where d_c is the number of objectives which are either maximized or minimized. Denote each component of f by $f_i(\theta)$, and without loss of generality, assume that our goal is to minimize all components; then the MOO problem can be written as

$$\begin{aligned} &\min_{x \in \Theta} \left(f_1(\theta), ..., f_{d_c}(\theta)\right) \\ &\text{s.t. } g(\theta) = 1 \end{aligned} \tag{4.1}$$

where $g : \Theta \to \{0, 1\}$ represents a constraint function. Then the *feasible domain* is the set of points in input space which respect the constraint, $\bar{\Theta} := \{\theta : g(\theta) = 1, \forall \theta \in \Theta\}$. The *attainable set* is the image by f of the *feasible domain* $\bar{\mathcal{C}} := \{f(\theta) : \forall \theta \in \bar{\Theta}\}$. In most applications, it is not possible to simultaneously minimize all objectives as they often "conflict" with each other, which means that an improvement in one objective deteriorates other objectives (otherwise the MOO problem would be degenerated and therefore equivalent to a single objective problem). Therefore, the **solution to (4.1) is a** *set* **of possibly infinite cardinal**. Because these solutions may be incomparable, the solution set is defined via partial ordering as opposed to total ordering (as in the single-objective case). To simplify the notation we will denote $f(\theta) := c$ and $f_i(\theta) := c_i$.

Definition 4.1 (Dominance partial ordering)**.** *For two points in the attainable set, $c^{(1)}, c^{(2)} \in \bar{\mathcal{C}}$, $c^{(1)}$ is said to* **dominate** *$c^{(2)}$, written $c^{(1)} \prec c^{(2)}$ if and only if for all $i \in [1, d_c]$ $c_i^{(1)} \leq c_i^{(2)}$ and, there exist $j \in [1, d_c]$ such that $c_j^{(1)} < c_j^{(2)}$.*

With this definition, it is possible to define the notion of *Pareto optimality*.

Definition 4.2 (Non Dominance)**.** *A point of the attainable set $c^* = f(\theta^*)$, $\theta^* \in \bar{\Theta}$ is said to be* **non-dominated** *if and only if $c \not\prec c^*$ for all $c \in \bar{\mathcal{C}}$. The corresponding θ^* is said to be* **efficient**.

Then the solution to (4.1) is called the *Pareto-Frontier* (PF), which is the set of all non-dominated points formally defined as $\mathcal{F} := \{c^* : c^* = f(\theta^*) \text{ is not dominated }\}$. The corresponding set of all efficient points θ^* is called the *Pareto-Set* (PS).

In general, when all objectives are conflicting and continuous, the PF is a $(d_c - 1)$-dimensional trade-off surface embedded in $\mathcal{C}$ (*e.g.*, with 2 objectives the PF is a curve). To read more about MOO definitions and terminology, see Ehrgott (2005, Chapters 1 and 2).

4.2.1 . Metrics of performance in multi-objective optimization

As in our set of methods of interest, it is not possible to return an infinite set of solutions to cover the PF, we must approximate it via a discrete set instead. Therefore a central challenge of MOO research is to measure the quality of the approximation of the PF.

To evaluate how well a discrete set of approximate solutions describe the PF's true shape is an open problem, and many metrics of MOO performance have been proposed (Audet et al., 2021). One desirable property of a MOO performance indicator is *Pareto compliance*. Let A and B be the sets of (approximately) non-dominated solutions returned by two different algorithms. Then $A \prec B$ if for every $b \in B$, there exists $a \in A$ such that $a \prec b$. An indicator I is *Pareto compliant* if either $A \prec B$ implies that $I(A) < I(B)$ or $I(A) > I(B)$ (depending on whether the indicator is increasing or decreasing with improved quality). To our knowledge, the only MOO performance indicators that possess this property are the **hypervolume indicator** (HVI) and the **improved inverse generational distance** (IGD^+) (Ishibuchi et al., 2015). The **improved generational distance** (GD^+) will also be of interest to us, although it does not possess this property. All of these methods suffer from one drawback in that they rely on an appropriate choice of one or more reference points, which may require *a priori* knowledge of the true PF.

The HVI is given by $HVI(A) = V(\cup_{a \in A}[a, c_{\text{ref}}])$, where $V(\cdot)$ denotes the volume, $[a, c_{\text{ref}}]$ denotes a hypercube with lower bound a and upper bound c_{ref}, and c_{ref} denotes the reference point, which must be dominated by every solution point (*a.k.a.*, the *Nadir point*). As mentioned above, the HVI is Pareto compliant. In practice, it is typically possible to select the reference point for the HVI by using unacceptably bad scores for each objective. However, an overly poor choice of reference points can lead to non-interpretable large values. The HVI is also extremely sensitive to poor problem scaling. Finally, it is worth noting that HVI is exponentially expensive to calculate when $d_c > 2$ (Ishibuchi et al., 2015).

The IGD^+ and GD^+ indicators are defined in terms of a modified distance metric $d^+(\hat{c}, c) := \| \max(\hat{c} - c, 0) \|_2$, where c is a target point, $\hat{c}$ is an objective point in the estimated PF, and the $\max$ is taken element-wise. Then given a set of target points $\mathbf{C}$ and a set of estimated points on the PF $\hat{\mathbf{C}}$, the GD^+ indicator is given by:

$$\text{GD}^+(\hat{\mathbf{C}}; \mathbf{C}) := \sum_{\hat{c} \in \hat{\mathbf{C}}} \min_{c \in \mathbf{C}} d^+(\hat{c}, c) / |\hat{\mathbf{C}}|$$

and the IGD^+ indicator is given by:

$$\text{IGD}^+(\hat{\mathbf{C}}; \mathbf{C}) := \sum_{c \in \mathbf{C}} \min_{\hat{c} \in \hat{\mathbf{C}}} d^+(\hat{c}, c) / |\mathbf{C}|$$

(with $|.|$ the cardinal operator). In practice, these indicators are faster to compute than HVI. However, the need for a large target set $\mathbf{C}$ that "covers" the entire PF can be impossible to satisfy when the true PF is unknown, making these indicators difficult to use in real-world applications.

Balancing solution *quality* (*i.e.*, closeness between approximated and true PF) against *diversity* (*i.e.*, coverage of the approximated PF) is considered one of the central challenges

of MOO (Audet et al., 2021; Deb et al., 2002). While HVI and IGD$^+$ are both Pareto compliant, they are redundant in that both tend to place a higher emphasis on diversity over quality. In this paper, we prefer using the HVI, which is more standard in the MOO literature. But, for algebraic (toy) problems, where the true PF is known, we also utilize GD$^+$ as a complementary metric, that places a higher emphasis on solution quality (*i.e.*, closeness of the estimated PF to the true PF).

4.2.2 . Scalarization of objectives

One classical approach to solving a multi-objective problem is to transform it into a single-objective problem by using a scalarization function $s_w : \mathcal{C} \to \mathbb{R}$ (Ehrgott, 2005, Chapters 3 and 4), parameterized by a weight vector $w \geq 0$ normalized to 1, which reflects the trade-off between objectives. The optimization problem now becomes

$$\begin{aligned} &\min_{x \in \Theta} \; s_w(f_1(\theta), \ldots, f_{d_c}(\theta)) \\ &\text{s.t. } \; g(\theta) = 1 \end{aligned} \tag{4.2}$$

For most common scalarizations s_w, each choice of w produces a different solution θ^* to (4.2), such that θ^* is efficient. By solving many instances of Problem 4.2 with different w, numerous solutions are produced giving an approximation to the PS. Many existing scalarization functions were proposed in the literature (Chugh, 2020), and one way to achieve parallelism is by solving numerous independent scalarizations (Chang and Wild, 2023b; Deb and Jain, 2013).

However, scalarization can be sensitive to scales and curvatures of objectives and often fails to produce complete coverage of the approximated PF. In particular, such characteristics can cause different trade-off parameters w to produce similar or identical solution points θ^*. This can result in grouped solutions on the approximated PF and may leave certain regions sparsely populated or empty. Such unbalanced and clustered coverage can give the user a biased understanding of the objectives trade-off.

To resolve such limitations it is possible to adaptively select weights such that the approximated PF has a better coverage (Das and Dennis, 1998; Deshpande et al., 2016). However, this can induce a sequential dependence between different instances of Problem 4.2, which can limit parallelism.

Alternatives to scalarization also exist and are often based on maximizing the HVI. In Bayesian optimization (BO), one would often maximize the expected improvement in the HVI in each iteration. The direct drawback of such an approach is the computational expense of calculating expected HVI improvement with more than two objectives, so one typically needs to approximate instead (Daulton et al., 2020). Another kind, more scalable, is based upon multi-objective generalizations of genetic algorithms, which sort points according to the non-dominance relation during the selection phase. The most popular in this class is the non-dominated sorting genetic algorithm (NSGAII) (Deb et al., 2002), which we use as a baseline in our experiments. One downside is that genetic algorithms can require several generations before they become significantly different from random sam-

pling, and many more to converge. An efficient parallelization through the island model also exists for NSGAII (Märtens and Izzo, 2013).

Finally, we introduce the set of scalarization functions we considered in our experiments. Let w be a normalized weight vector of dimension d_c. Let $c \in \mathcal{C}$ be an objective vector also of dimension d_c, and let z be the *utopia point* given by the objectives minimized independently.

The weighted-sum or *linear* (L) scalarization is:

$$s_w^L(c) = \sum_{i=1}^{d_c} w_i c_i \tag{4.3}$$

The *Chebyshev* (CH) scalarization is:

$$s_{w,z}^{CH}(c) = \max_{i \in [1;d_c]} w_i |c_i - z_i| \tag{4.4}$$

The *penalty-boundary intersection* (PBI) scalarization is:

$$s_{w,z}^{PBI}(c) = d_1(z - c; w) + \lambda \cdot d_2(z - c; w) \tag{4.5}$$

where $d_1(c; w) = |c^\top w / \|w\||, d_2(c; w) = \|c - d_1 w / \|w\|\|$ and $\lambda \in \mathbb{R}$ a parameter (default to 5).

4.2.3 . Review of multi-objective hyperparameter optimization methods

In HPO, Θ is often a mixed-integer search space composed of categorical, discrete, and continuous variables (Section 2.1). The target objectives are non-smooth. The optimized workflow is expensive to evaluate (in memory, in time). The run-time of evaluated workflows depends on the hyperparameters and has variability. Also, failures (*a.k.a.*, hidden constraints) can appear for some hyperparameter configurations (*e.g.*, out-of-memory, neural network training resulting in `NaN`).

To our knowledge, very few open-source software have features that can handle all these requirements and also perform asynchronous large-scale parallelism. One of the first solvers to offer support for multi-objective BO is the Pareto Efficient Global Optimization (ParEGO) software, which uses augmented Chebyshev scalarization followed by a Gaussian process surrogate model and expected improvement maximization to select the next iterate(s) (Knowles, 2006; Cristescu and Knowles, 2015). Some newer notable software include pymoo (Blank and Deb, 2020) (official implementation of NSGAII and other genetic algorithms), BoTorch with the quasi-expected HVI (qEHVI) acquisition (Balandat et al., 2020; Daulton et al., 2020) (Bayesian optimization with approximate expected HVI maximization), ParMOO (Chang and Wild, 2023b) (customized MOO algorithms), DeepHyper (Balaprakash et al., 2018a) (parallel AutoML for HPC), Optuna (Akiba et al., 2019a) (HPO solvers). More details about some software features can be found in Chang and Wild (2023a, Table 1) and Karl et al. (2022). In this study, we will focus on DeepHyper for its asynchronous BO solver; Optuna for its asynchronous NSGAII, MoTPE, and BoTorch+qEHVI implementations (Ozaki et al., 2022, 2020); the ParEGO implementation from the SMAC Python package (Lindauer et al., 2022) that provides a Random-Forest surrogate model for comparison with ours.

Algorithm 3: Asynchronous Decentralized Multi-Objective Bayesian Optimization (Worker Process)

Inputs : `thetaSpace`: a configuration space,
`nInitial`: the number of initial hyperparameter configurations,
`f`: a function that returns the cost of the learning workflow,
κ: the parameter of qLCB,
T: the period of the exponential decay,
λ: the decay rate of the exponential decay,
`nObj`: the number of objectives to optimize,
`upperBoundObj`: the objectives upper bound constraints,
γ: the penalty strength for objective bounds (defaults to 2)

Output: `thetaStar` the recommended Pareto-set of hyperparameter configuration.

`thetaArray`, `costArray` ← New empty arrays of hyperparameter configurations and costs ;
`model` ← New RFR surrogate model ;
$\kappa_0 \leftarrow$ Sample from exponential distribution with parameter $1/\kappa$;
$t \leftarrow 0$;
while *compute time is not exhausted* **do**
    /* Apply exponential decay */
    $\kappa_t \leftarrow \kappa_0 \cdot \exp(-\lambda \cdot ((t - \texttt{nInitial}) \bmod T))$;

    if *Length of* $\texttt{thetaArray} < \texttt{nInitial}$ **then**
        `theta` ← Sample hyperparameter configuration from `thetaSpace` ;
    else
        /* Quantile-Uniform Normalization */
        $\hat{F} \leftarrow$ Estimate empirical cumulative distribution function from `costArray` ;
        $\texttt{costArrayQU} \leftarrow \hat{F}(\texttt{costArray})$;
        /* Apply Penalty */
        $\texttt{upperBoundObjQU} \leftarrow \hat{F}(\texttt{upperBoundObj})$;
        $\texttt{penalty} \leftarrow \gamma \sum_{i \in [|1, \texttt{nObj}|]} \max(\texttt{costArrayQU}[:, i] - \texttt{upperBoundObjQU}[i], 0)$;
        $\texttt{costArrayPenalized} \leftarrow \texttt{costArrayQU} + \texttt{penalty}$;
        /* Scalarization */
        `weights` ← Sample array of `nObj` weights from $\Delta\texttt{nObj}$;
        `costArrayScalarized` ← Apply scalarization $s_{\texttt{weights}}(.)$ on all cost vectors in `costArrayPenalized` ;

        `costArrayAux` ← Returns new array where failures in `costArray` are replaced with $\max(\texttt{costArray})$;
        Update `model` with `thetaArray`, `costArrayAux` ;
        `theta` ← Returns `theta` in `thetaSpace` that minimizes $a_{\text{LCB}}(\theta; \kappa_t)$ for current `model` (random sampling or genetic algorithm) ;
    end
    `cost` ← Returns the cost of learning workflow `f(theta)` ;

    /* Exchange observations */
    `thetaArrayNew`, `costArrayNew` ← Collects new observations from other workers ;
    Share new observation `theta`, `cost` with other workers ;

    `thetaArray`, `costArray` ← Concatenate `thetaArray` with `thetaArrayNew` and `costArray` with `costArrayNew`;
    `thetaArray`, `costArray` ← Concatenate `thetaArray` with [`theta`] and `costArray` with [`cost`];
    `thetaStar` ← Update recommendation ;
    $t \leftarrow t + 1$;
end

4.3 . Parallel multi-objective Bayesian optimization with uniform normalization and bounded objectives

In this section, we propose our decentralized multi-objective Bayesian optimization (D-MoBO) algorithm. It combines a parallel decentralized asynchronous architecture with a sequential MoBO algorithm.

4.3.1 . Decentralized Bayesian optimization

The asynchronous decentralized scheme we use was presented in Chapter 3.3 (Egelé et al., 2023). The main idea is to start independent Bayesian optimization agents in parallel. Each agent is running a sequential BO algorithm but stores its observations in a shared memory space (*e.g.*, database). The trick is to initialize different exploitation-exploration parameters combined with an exponential-decay scheduler on this parameter to avoid "over"-exploring when increasing workers (Algorithm 2).

4.3.2 . Multi-objective Bayesian Optimization

The multi-objective Bayesian optimization (MoBO) algorithm we propose is presented in Algorithm 3. It is inspired by the ParEGO (Knowles, 2006) algorithm which performs scalarization through the augmented Chebyshev function and the SMAC algorithm (Hutter et al., 2012) which uses a Random-Forest (RF) surrogate model with random-splits. A similar variant is already available in the SMAC3 (Lindauer et al., 2022) package.

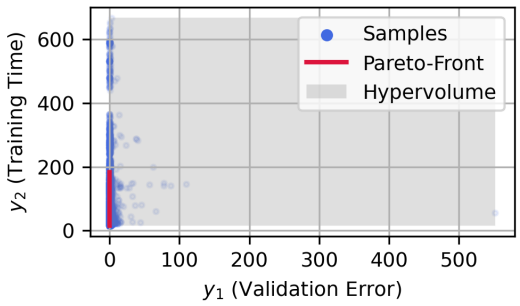
Uniform normalization of objectives

(Lines 11-12, Algorithm 3) In the HPO setting, objectives of interest can have *different scales* (*e.g.*, accuracy, latency, FLOPS). Not only that but *outliers* are also common when exploring a large hyperparameter search space (*e.g.*, diverging metrics, numerical errors), see Figure 4.1a which display typical observations from MOO on the NavalPropulsion task from HPOBench (Klein and Hutter, 2019b; Eggensperger et al., 2021b) (other tasks of the considered benchmarks displayed similar behavior). On the one hand, the combination of both effects can make the HVI computation highly non-trivial. For example, the objective with the largest scale can weigh excessively and hide improvements in other objectives. Also, choosing a normalization and a reference point can become dependent on the experiment due to sensitivity to outliers such as in Figure 4.1a where most of the HVI between the top-right corner of the figure and the PF (red line) is empty. On the other hand, the RF model used for BO can typically "under-fit" optimal configurations which often have the smallest squared error (*e.g.*, closer to zero) already in the single-objective optimization setting. Previous work on single-objective optimization (Egelé et al., 2023; Lindauer et al., 2022), usually apply some sort of "log"-based transformation to mitigate the under-fitting problem. For instance, the MinMax-Log transformation $t^{\mathsf{MML}}(c) = \log\left((c - c_{\mathsf{min}})/c_{\mathsf{max}} + \epsilon\right)$ is usually effective. However, we noticed that such transformation is also sensitive to outliers in the multi-objective case and while it was not a problem for single-objective, it can transform a convex PF into a non-convex which makes the MOO

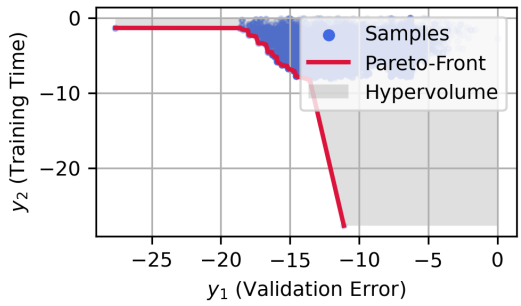
problem much harder such as in Figure 4.1b.

For these reasons, as discussed in Section 4.2.2, scalarization can perform poorly on real-world problems. In particular, it is well-known that uniformly sampled weights do *not* produce uniformly distributed solution points on the PF, especially for linear scalarization (Das and Dennis, 1998). To resolve these problems, we require a transformation that would conserve the PF properties, focus on areas of interest (*i.e.*, close to the estimated PF), and be robust to outliers. A mapping of the independent objective distributions $P(c_i)$ to the uniform distribution can provide such properties. This also helps apply randomized scalarization without worrying about differences in objective scales and curvatures.

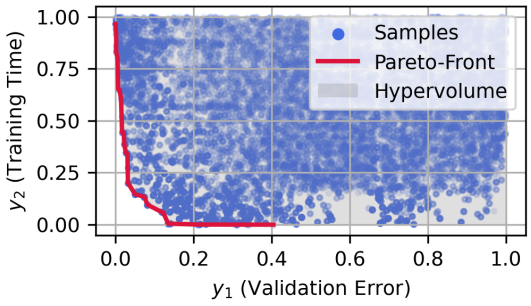
To do so, following (Amaratunga and Cabrera, 2001) and (Bolstad et al., 2003) we perform quantile-normalization by composing the empirical cumulative distribution function (ECDF) $\hat{F} : \mathbb{R} \to [0, 1]$ with the quantile function (*i.e.*, inverse of CDF) of the uniform distribution $Q_{\mathcal{U}(0,1)} : [0, 1] \to \mathbb{R}$. As the latter is the identity function $Q_{\mathcal{U}(0,1)}(\theta) = \theta$ we just need to apply $\hat{F}(\theta)$. This means that we map each objective to a uniform distribution on $[0, 1]$. This allows us to have a better update of the surrogate model. The ECDF is estimated from the observed objectives Υ. The quantile-uniform (QU) transformation is $t^{\text{QU}}(c) = \hat{F}(c)$. The result of this transformation is illustrated in Figure 4.1c. To show that



(a) Identity (Id)

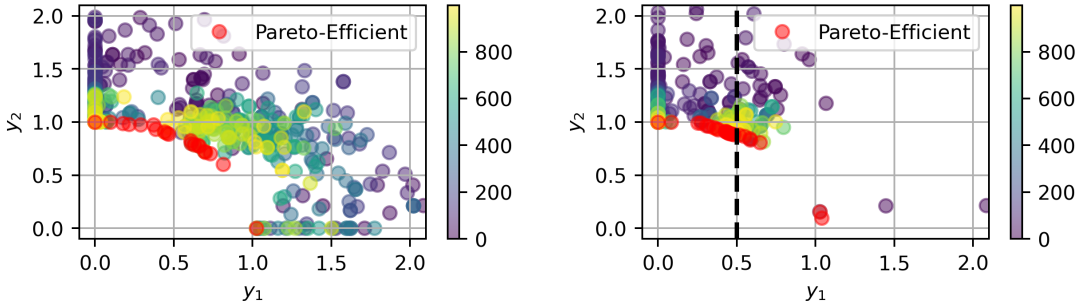


(b) MinMax-Log (MML)



(c) Quantile-Uniform (QU)

Figure 4.1: Comparing normalization of objectives on a 2-objectives hyperparameter optimization instance (NavalPropulsion from HPOBench) where both validation error c_1 and training time c_2 are minimized.



(a) No Penalty (NP)

(b) Penalty $c_1 < 0.5$

Figure 4.2: Example effect of the penalty on the DTLZ 2 benchmark (Deb et al., 2005). The color of points indicates the BO iteration at which the points were evaluated.

the above transformation preserves Pareto optimality, *i.e.*, c is non-dominated in $\bar{\mathcal{C}}$ if and only if $t^{QU}(c)$ is non-dominated in $t^{QU}(\bar{\mathcal{C}})$, consider the following claim.

Claim 1 (Invariance under order-preserving transformations)**.** *If $t : \mathbb{R}^{d_c} \to \mathbb{R}^{d_c}$ is a componentwise order-preserving transformation on $c \in \bar{\mathcal{C}}$ such that $t_i(c^{(j)}) < t_i(c^{(k)}) \Rightarrow t_i(c^{(j)}) < t_i(c^{(k)})$ and vice versa, then $t(c^*)$ is nondominated in $t(\bar{\mathcal{C}})$ if and only if c^* is nondominated in $\bar{\mathcal{C}}$.*

Proof. Suppose that t is order-preserving, as defined above.

($\Rightarrow$): Assume that $t(c^*)$ is Pareto optimal in $t(\bar{\mathcal{C}})$, which implies that $c^{(t,j)} \not\prec t(c^*)$ for all $c^{(t,j)} \in t(\bar{\mathcal{C}})$. For contradiction, suppose that $c^{(j)} \prec c^*$ for some $c^{(j)} \in \bar{\mathcal{C}}$. Then by definition $c^{(j)} \leq c^*$ and $c_i^{(j)} < c_k^*$ for at least one $i \in \{1, \ldots, o\}$. Since t is order-preserving, this would imply that $t_i(c^{(j)}) < t_i(c^*)$ and componentwise $t(c^{(j)}) \leq t(c^*)$. Since $t(c^{(j)}) \in t(\bar{\mathcal{C}})$, this is a contradiction.

($\Leftarrow$): Assume that c^* is Pareto optimal in $\bar{\mathcal{C}}$, which implies that $c^{(j)} \not\prec c^*$ for all $c^{(j)} \in \bar{\mathcal{C}}$). Each $c^{(t,j)} \in t(\bar{\Theta})$ satisfies $c^{(t,j)} = t(c^{(j)})$ for some point in the untransformed set $\bar{\Theta}$. But from the assumption, $c^{(j)} \not\prec c^*$. So by similar logic as above, $t(c^{(j)}) \not\prec t(c^*)$. □

Corollary 1 (Quantile Uniform Transformation)**.** *The quantile transformation t^{QU} is an ECDF. By definition, an ECDF is monotone increasing (strictly monotone when invertible), so it is immediately order-preserving. So from the claim, we immediately conclude that t^{QU} preserves the Pareto set.*

Penalty on constrained objectives

(Lines 10-15, Algorithm 3) Even though we are interested in exploring a diverse set of solutions on the PF, there are often minimal requirements on some objectives. For example, if we have a baseline binary classifier with an error rate of 15%, likely, we would likely not consider any solution configuration with an error rate greater than 20%. In the generic

multi-objective Problem 4.1, there is no way to specify that some solutions are fundamentally less interesting than others. Therefore, we propose to impose upper bounds on the objective ranges. Since these constraints could be violated (*i.e.*, obtaining some results out of bounds), we can consider these objective bounds as *soft* constraints (Le Digabel and Wild, 2015).

In the single-objective literature, nonlinear constraints are often handled via a *penalty function*, such as an augmented Lagrangian. This technique generalizes to the multi-objective case, where the penalty must be applied to all objectives (not only the objective that violates its upper bound) (Cocchi and Lapucci, 2020). In our case where the constraint functions are also black-box functions, the *progressive barrier* penalty function has been shown to work well in the single-objective case (Audet and Dennis, 2009). In other multi-objective black-box software, the progressive boundary approach has been successfully implemented in the multi-objective case and shown to be effective in handling arbitrary black-box constraints (Chang and Wild, 2023a).

To enforce upper bounds on objective ranges, we apply a progressive barrier penalty to all objectives whenever one or more objectives violate their upper bounds. The penalty is calculated as the sum of all constraint violations multiplied by a penalty strength factor γ. See the exact calculation in Algorithm 3. Note that, thanks to the QU transformation, it is appropriate to choose a problem-independent penalty constant of $\gamma = 2$, which is slightly greater than the normalized objective magnitudes. This penalty discourages the optimizer from wasting resources further refining uninteresting trade-offs, which fail to meet the minimal requirements. An example of the effect of such a penalty is provided in Figure 4.2. In the case where no penalty is applied (Figure 4.2a) the full PF of the problem is explored until the end (yellow points). But, when applying the penalty to enforce $c_1 < 0.5$ (Figure 4.2b) we observe that most of the PF where $c_1 > 0.5$ is rarely explored.

We stress two limitations of such a penalty. First, when the true PF is not known using a penalty can be ineffective. Indeed, if the penalty does not impact the PF then we observed that the performance of the MOO algorithms is similar or worse than not having the penalty. Second, this penalty strategy is particularly effective for QU normalization. However, when using the same scheme with other transformations (Identity and MinMax-Log) it was very much ineffective or required harder tuning of the γ parameter.

Randomly weighted scalarization of objectives

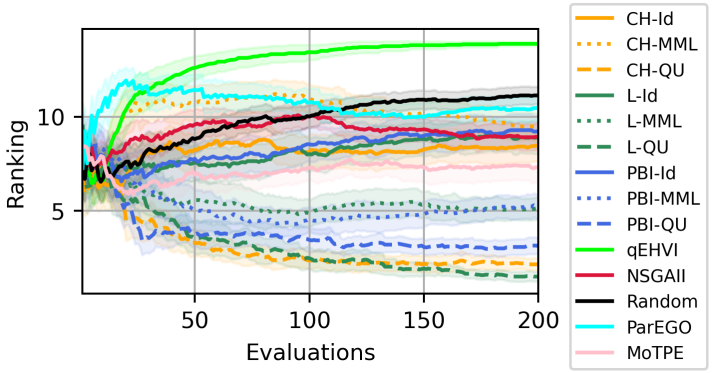
(Lines 16-17, Algorithm 3) We focus on scalarization-based MoBO and as it is not clear which function is better we consider several functions from the literature (Chugh, 2020): weighted-sum or *linear* (L), the *Chebyshev* (CH), and the *penalty-boundary intersection* (PBI) (Section 4.2.2). Then, to enhance the diversity of the estimated PF, in each BO iteration we decide to re-sample weights uniformly from the unit-simplex Δd_c (Pinelis, 2019, Remark 1.3) $w_i = \log(1 - \tilde{w}_i)/(\sum_{j=1}^{d_c} \log(1 - \tilde{w}_j))$ with $\tilde{w}_i \sim \mathcal{U}(0, 1)$.

4.4 . Experimental results

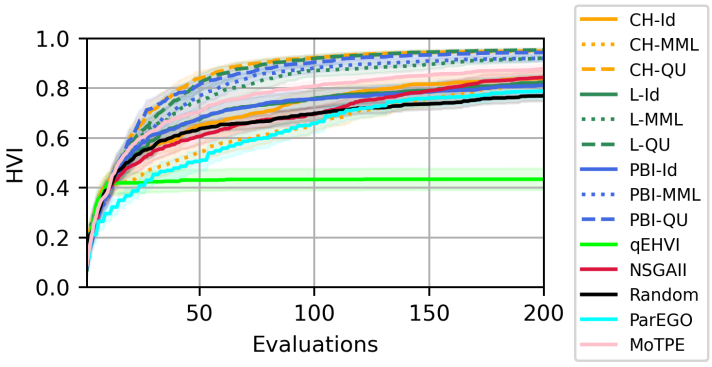
In this section, we present our experimental results on D-MoBO (Algorithm 3). First, we evaluate which combination of scalarization function and normalization is the most effective while also comparing it to other state-of-the-art algorithms. Then, we evaluate the impact of the penalty on the optimization process. Finally, we evaluate the gain in performance of D-MoBO when scaling parallel workers.

4.4.1 . Evaluation of different scalarization and normalization

In this section, we present the experimental results which led us to choose the Linear scalarization with QU normalization. For this, we select four HPO benchmark tasks from HPOBench (Eggensperger et al., 2021b; Klein and Hutter, 2019b): NavalPropulsion, ParkinsonsTelemonitoring, ProteinStructure, SliceLocalization. The two objectives are the validation error and the training time. These benchmarks allow for fast evaluations of HPO methods as they simulate learning workflow evaluations through a pre-computed database. Also, in this experiment, we limit ourselves to studying the behavior of Algorithm 3 for 200 sequential iterations. For the comparison, we choose three scalarization functions: Linear (L), Chebyshev (CH), and PBI. And, three objective normalization strate-



(a) Ranking



(b) Hypervolume

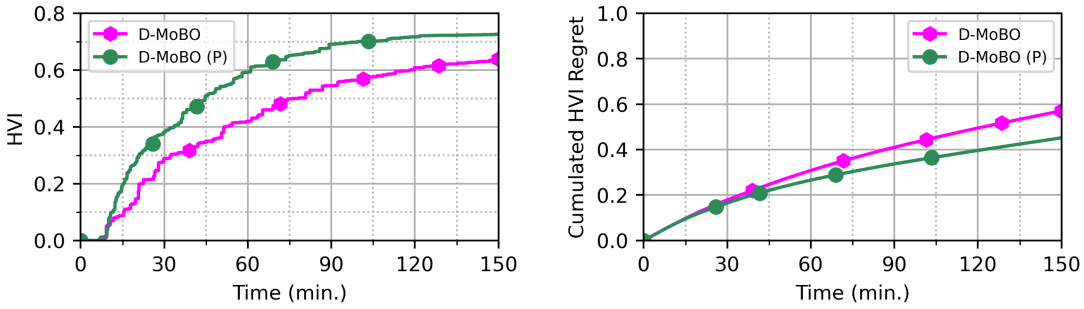
Figure 4.3: Comparing scalarization functions and objective normalization combinations on the HPOBench tabular tasks (*i.e.*, 40 experiments per curve).

gies: Identity (Id), MinMax-Log (MML), and Quantile-Uniform (QU). As we want to study the synergy between scalarization and objective normalization each combination (9 in total) is evaluated. We add five baselines for the comparison: random (Random), NSGAII, MoTPE, ParEGO, and BoTorch with the qEHVI implementation of expected HVI (qEHVI). Each configuration is repeated 10 times with fresh random states. The quality metric is HVI. In Figure 4.3a, we present the averaged ranks over HVI curves across all tasks and repetitions (*i.e.*, 40 experiments). The average HVI curves are also given in Figure 4.3b. Scalarization functions have different colors and objective normalization has different line styles. The transparent bandwidth around the curves represents a confidence interval of 95% confidence of the mean-value estimation s (*i.e.*, 1.96 standard error).

From the ranking curves (Figure 4.3a) (lower ranks are better), it is clear that **QU normalization is the best** as strategies based on QU (dashed lines) have curves which dominate all other methods independently of the scalarization function. Then for the scalarization functions, it appears that PBI (blue dashed line) is under-performing compared to L (green dashed line) and CH (orange dashed line). As L is winning we keep it as the default scalarization function. However, because L and CH are statistically different (*i.e.*, overlapping standard errors) and also because we know that CH would be more robust in the case of non-convex PF we keep it as an option for future work. Indeed, CH may be harder to optimize for MoBO and could require a few iterations with fixed weights instead of always re-sampling weight vectors. Finally, from 100 evaluations it seems like NSGAII manages to slowly improve its ranking which seems normal as its population size is 50. The qEHVI under-performs on these 4 tasks probably because the surrogate model is a Gaussian process while the search space does not have any continuous ranges. The ParEGO implementation from SMAC which is also based on a Random-Forest surrogate model slightly outperforms Random and is close in ranking to our variants with Identity (Id) objective scaling which puts forward the advantage of objective scaling. Interestingly, the MoTPE is performing better than any of the BO methods with Identity (Id) objective scaling.

4.4.2 . Adding the penalty to avoid uninteresting candidates

In this section, we show the effect of the penalty to explore only "useful" hyperparameter configurations. For this, we use a different benchmark more suitable for the HPC setting. We choose the Combo problem from the ECP-Candle benchmark which was introduced in previous works for AutoML on HPC (Balaprakash et al., 2019; Égelé et al., 2021; Egelé et al., 2023). This benchmark contains 22 hyperparameters and performs a regression task on the growth rate of Cancer cells given a treatment. The multi-objective problem is to minimize $c_1 := 1 - R^2$ (*i.e.*, maximizing R^2 the coefficient of determination on validation data), minimize c_2 the latency (*i.e.*, inference time), and minimize the number of parameters (*i.e.*, model size). Each model can train for a maximum of 50 epochs and 30 minutes. Experiments are run on a maximum of 640 GPUs in parallel (1 evaluation per GPU) for 2.5 hours (*i.e.*, corresponds to only 5 sequential evaluations of 30 minutes). Only 1 repetition is done per experiment due to their cost and the random seed is fixed to 42 for all experiments. The penalty upper-bound is set to $c_1 < 0.15$ as the baseline model from Combo reaches $c_1 = 0.13$ after 100 epochs.



(a) Hypervolume

(b) Cumulated Hypervolume Regret

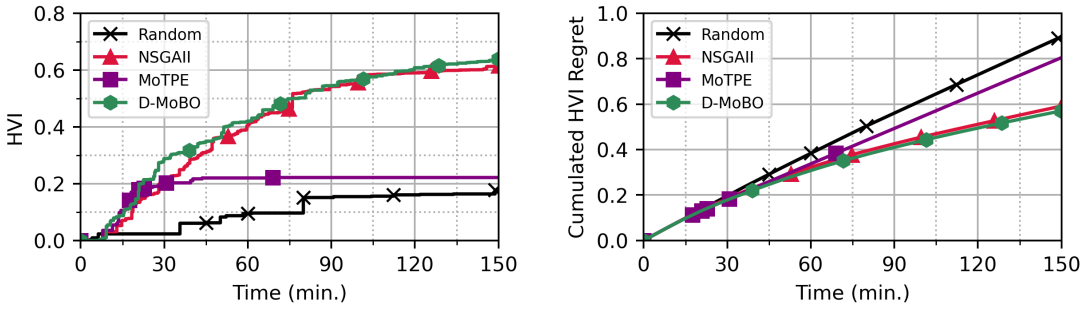
Figure 4.4: Observing the effect of penalty for D-MoBO on the Combo benchmark. The penalty is set to $R^2 > 0.85$. D-MoBO (green), is the version implementing the penalty while D-MoBO (NP) (pink) is without penalty. HVI is computed using the reference for $R^2 = 0.85$.

In Figure 4.4a we present the HVI vs time curve for D-MoBO (without penalty), and D-MoBO (P) (with penalty). The reference point to compute the HVI is set $c_{\text{ref}} = (0.15, \max(c_2), \max(c_3))$. Such a penalty represents the practical consideration where any model with a predictive performance less than a threshold is unusable (minimum requirement). It is clear from the results that applying the penalty helps the algorithm focus on the solution set of interest as the HVI curve of D-MoBO (P) increases much faster than D-MoBO. Similar improvements in the results were observed with fewer parallel workers (40 and 160). Then in Figure 4.4b we show the normalized temporal cumulated HVI regret curves. We define the normalized temporal cumulated HVI regret by normalizing the temporal cumulated regret (Definition 2.4) according to the maximum time.

$$R(t_i) \approx \sum_{i'=0}^{i} (1 - HVI(A_{i'})) \cdot \frac{(t_{i'+1} - t_{i'})}{t_{\text{max}}} \tag{4.6}$$

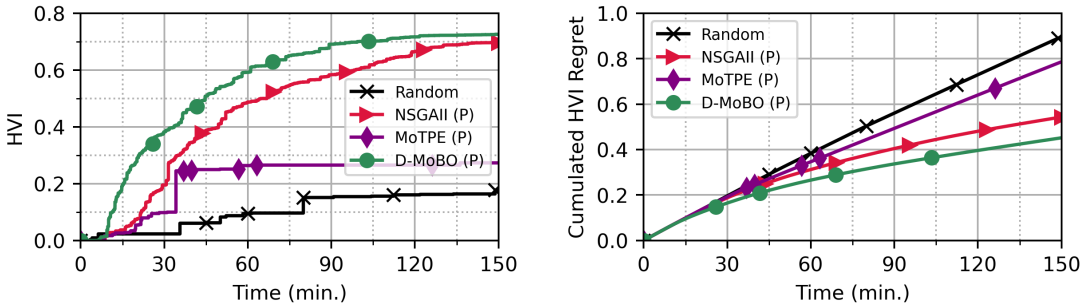
where t_i is the time at which the i-th evaluation was completed, $1 - HVI(A_{i'})$ is hypervolume regret after the i-th evaluation was completed, and t_{max} is the maximum time (in our case 2.5 hours). The cumulated regret quantifies both the quality of the solution and the convergence rate as detailed in (Srinivas et al., 2010). A linear cumulated regret often corresponds to a uniformly random strategy (*e.g.*, Random in Figure 4.5b). The algorithm that has the best convergence rate (*i.e.*, how fast good solutions are good) will be lower than others. Figure 4.4b shows that using the penalty improves the convergence rate. Also, in Table 4.1 we provide additional quantitative metrics. Mainly, when comparing D-MoBO and D-MoBO (P) we notice that without the penalty very few models are reaching the threshold of required accuracy (denoted by %V). For example, the penalty helps, for the same computational budget (640 workers), to move from 12.60% to 67.21% of successfully evaluated models with $c_1 < 0.15$.

Then, we also compare D-MoBO with other noticeable MOO algorithms of the literature (Figure 4.5): NSGAII, MoTPE (both from Optuna, Akiba et al. (2019b), with default



(a) Hypervolume (b) Cumulated Hypervolume Regret

Figure 4.5: Comparing D-MoBO against other optimizers (all without penalty) on the Combo benchmark.



(a) Hypervolume (b) Cumulated Hypervolume Regret

Figure 4.6: Comparing D-MoBO (P) against other optimizers (all with penalty) on the Combo benchmark.

parameters), and Random search. While Random search is mostly used as a sanity check of HPO algorithms in general, NSGAII is known to be a strong performer in the MOO field and MoTPE is known to have faster convergence. Here we do not compare to ParEGO or qEHVI for two reasons. First, they did not perform well in the first set of experiments and second, because we do not have a scalable implementation of ParEGO, and qEHVI does not scale to our setting even if it is available in Optuna, with which we run NSGAII and MoTPE, as it is based on a Gaussian process surrogate model with cubic temporal complexity in the number of observations (see Optuna developers (2018)). In Figure 4.5 we observe that D-MoBO and NSGAII have very similar performance for both the HVI and the regret. However, we can see in Figure 4.5a that Random and MoTPE are performing significantly worse. MoTPE improves significantly over Random and slightly better than NSGAII in the first half (< 30 minutes), and then stagnates.

Finally, we compare D-MoBO (P) with NSGAII (P) and MoTPE (P) also enforcing the constraint $c_1 < 0.15$ (Figure 4.6). In Figure 4.5a, we can observe that D-MoBO (P) has the advantage in early iterations but NSGAII (P) is closing the gap in late iterations. This effect

	40 Workers					160 Workers					640 Workers				
	#D	#F	%V	↑ HVI	↓ R_{final}	#S	#F	%V	↑ HVI	↓ R_{final}	#S	#F	%V	↑ HVI	↓ R_{final}
Random	545	8	0.74	0.13	0.94	2,115	42	0.53	0.11	0.94	8,418	178	0.36	0.28	0.90
MoTPE	219	12	0.48	0.05	0.95	1,129	2	1.86	0.12	0.91	5,571	13	2.25	0.22	0.80
NSGAII	496	4	2.85	0.25	0.90	2,050	7	8.96	0.44	0.76	10,092	14	22.96	0.61	0.59
D-MoBO	311	2	1.94	0.21	0.93	1,105	5	5.18	0.40	0.79	4,620	18	12.60	0.64	0.57
MoTPE (P)	312	2	0.0	0.0	1.00	1,792	4	15.21	0.13	0.94	4,866	136	8.58	0.27	0.79
NSGAII (P)	693	3	22.32	0.22	0.96	3,508	4	58.70	0.54	0.70	13,056	19	71.02	0.70	0.54
D-MoBO (P)	496	0	49.19	**0.36**	**0.76**	1,719	5	60.91	**0.68**	**0.52**	6,805	16	67.21	**0.73**	**0.45**

Table 4.1: Comparing optimizers for different numbers of workers. #D: the number of evaluations performed (successful or with failure), #F: the number of failed evaluations, %V: the percentage of successful evaluations with valid objective bounds, HVI: the final HVI, and R_{final}: the final cumulated HVI regret.

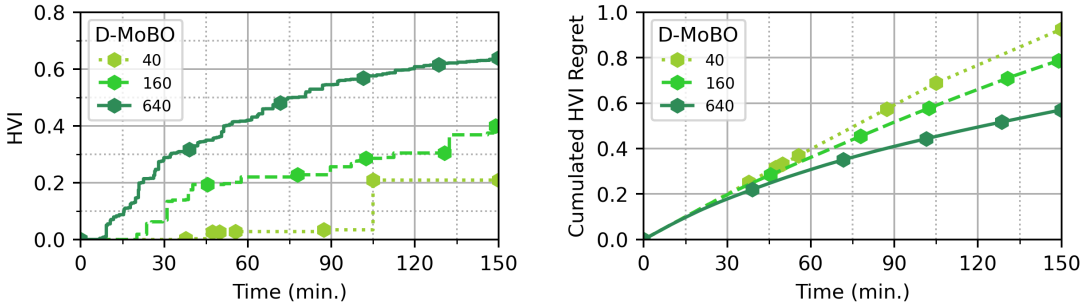
is confirmed with different numbers of workers, see the appendix. We can also see that MoTPE (P) stagnates quickly to barely outperform the Random search. MoTPE was not expected to scale so poorly (here using 640 parallel workers). Then, in Figure 4.6b we see through the cumulated regret curves D-MoBO (P) has the best convergence rate, closely followed by NSGAII (P), then comes MoTPE (P), and Random search.

4.4.3 . Improved optimization when scaling parallel workers

In this section, we show the gain in solution quality and convergence rates when increasing the number of parallel workers for D-MoBO and D-MoBO (P). For this, we follow the same experimental setting as the previous section. In addition to the experiments with 640 GPUs, we also run D-MoBO with 160 and 40 GPUs (*i.e.*, varying the workers by a factor of 4). In Figure 4.7 and 4.8, we see that increasing workers improves both solution quality and convergence rate (*i.e.*, how fast we get to the solution). For example, looking at the convergence speed of D-MoBO (P), with 640 workers in 30 mins we reach the final solution of 40 workers in 2.5 hours (*i.e.*, 5x speed-up). The cumulated regret curves also show consistent improvements in convergence rates. Then, for the solution quality, if we look at any point in time the experiments with more workers have larger HVI. To aggregate these, we provide, in Table 4.1, the HVI indicator at the end of the experiment (HVI, a metric of solution quality) and the cumulated regret also at the end of the experiment R_{final} (convergence rate) for each scale of workers. Both are improving with increased workers.

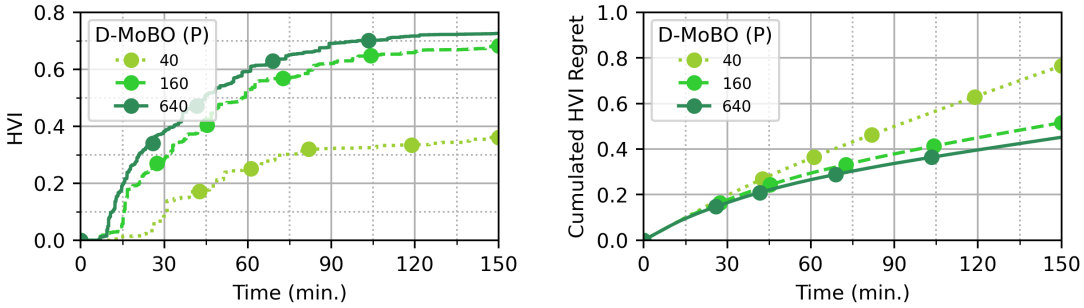
Similar scaling experiments were performed for other MOO algorithms: NSGAII, MoTPE, and Random search. We notice that NSGAII (the strongest competitor in our study) gains similarly in performance when increasing the number of parallel workers. However, for MoTPE and Random search, the gain in performance is minor. Finally, we can see in Table 4.1 that D-MoBO (P) consistently outperformed other competitors at the different scales we tested. However, the difference in the final solution against NSGAII becomes less significant when the computational budget increases.

4.5 . Conclusion



(a) Hypervolume (b) Cumulated Hypervolume Regret

Figure 4.7: Observing the effect of increased parallel workers in D-MoBO (without penalty). From 40, 160 to 640 parallel GPUs. The more workers the better the solution and the convergence rate.



(a) Hypervolume (b) Cumulated Hypervolume Regret

Figure 4.8: Observing the effect of increased parallel workers in D-MoBO (P) (with penalty). From 40, 160 to 640 parallel GPUs. The more workers the better the solution and the convergence rate.

This study demonstrates that D-MoBO, both with and without penalty, serves as an effective approach for parallel multi-objective hyperparameter optimization (MOHPO). Its efficiency can be attributed to several factors: (1) its QU normalization, (2) its implementation of a soft penalty, and (3) its scalability, which allows it to leverage an increased number of parallel workers. A notable advantage of this method, as revealed by our findings, is its ability to consistently perform at or near the optimal level across various levels of parallelism. We term this capability "any-scale" performance. In practical terms, this implies that users need not meticulously select the number of parallel workers, as the quality of results will either improve or remain stable when additional computational resources are allocated.

However, we have observed that the disparity in solution quality (measured by hypervolume) between NSGAII and D-MoBO diminishes as the number of parallel workers increases. Additionally, we have noted that NSGAII consistently achieves a significantly higher number of successful evaluations compared to D-MoBO, while maintaining a similar level of worker utilization. This suggests that NSGAII exhibits a stronger inclination toward favoring hyperparameter configurations that are quick to train compared to D-MoBO. Interestingly, despite this bias, NSGAII's performance does not appear to suffer. Lastly, the objective bounds can be hard to set without prior information about the problem. However, for predictive accuracy, the optimal constant predictor can always be used as the worst objective bound.

Contrary to some previous benchmarks such as YAHPO-Gym (Pfisterer et al., 2022, Figure 4) we find that Bayesian optimization can significantly outperform other optimization algorithms. In addition, we observe that random search and MoTPE do not gain significantly from an increase in parallel computations even if a lot more evaluations were completed.

In future works, the policy to sample trade-off weights should be optimized as uniform sampling is simple but under-optimized and certainly not adapted to all considered scalarization functions. Also, the scalarization function study should be refined by testing more MOHPO problems to evaluate how frequent are non-convex PF. For this, we wish to use JAHSBench-201 (Bansal et al., 2022) and YAHPO-Gym (Pfisterer et al., 2022) which are two promising benchmarks for MOHPO currently being refined.

4.6 . Acknowledgement

This research was conducted in collaboration with Tyler Chang, Venkatram Vishwanath, and Prasanna Balaprakash. We would also like to thank Isabelle Guyon for participating with us in discussions that improved the quality of this work. This material is based upon work supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility. This material is based upon work supported by ANR Chair of Artificial Intelligence HUMANIA ANR-19-CHIA-0022 and TAILOR EU Horizon 2020 grant 952215.

5 - Early discarding at a constant epoch in hyperparameter optimization of neural networks

---

In the previous chapters, we have seen how to perform hyperparameter optimization of learning workflows for one (Chapter 3) or multiple (Chapter 4) objectives. However, this process is usually time-consuming due to repeated training. This is even more true in the case of deep neural networks. Therefore, early discarding techniques (such as SHA used in Chapter 3) limit the resources granted to unpromising candidates by observing the empirical learning curves and canceling training as soon as the lack of competitiveness of a candidate becomes evident. Nevertheless, little is understood about the trade-off between the aggressiveness of discarding and the loss of solution quality (e.g., predictive accuracy). In this chapter, we study this trade-off for several commonly used discarding techniques such as successive halving and learning curve extrapolation. Our surprising finding is that these commonly used techniques offer minimal to no added value compared to the simple strategy of discarding after a constant number of epochs of training. The chosen number of epochs mostly depends on the overall available compute budget. We call this approach i-Epoch (i being the constant number of epochs with which neural networks are trained) and suggest to assess the quality of early discarding techniques by comparing how their Pareto-Front (in consumed training epochs and predictive performance) complement the Pareto-Front of i-Epoch.

5.1 . Introduction to early discarding in hyperparameter optimization

Optimizing the configuration of a deep learning pipeline is a complex task that involves properly configuring the data preprocessing, training algorithm, and neural architecture. A configuration is a specification of so-called hyperparameters (Yu and Zhu, 2020), which control the behavior of pipeline elements and hence can greatly influence its final predictive performance. The objective is to identify the configuration of hyperparameters that achieves the best predictive performance, usually referred to as hyperparameter optimization (HPO).

As HPO is often done from a black-box optimization point of view, that is by observation of input configuration and output performance, a major challenge is the required computation to evaluate candidate hyperparameters by training deep neural networks. This greatly limits the number of testable hyperparameter configurations within a practical time frame. This is why multi-fidelity hyperparameter optimization with early discarding was proposed to switch the black-box problem to a "gray-box" optimization problem by observing the intermediate training performance of neural networks and using it as an estimate of the final performance. Such estimates can in principle be obtained at a computationally cheaper training stage and therefore save overall computation. In deep neural networks, the training epochs are usually used to perform early discarding. An epoch usually refers to making a full pass over the training data. The predictive performance versus the number of epochs is also known as a "learning curve" (Viering and Loog, 2022; Mohr and van Rijn, 2022).

HPO with early discarding trades-off computation with quality of extrapolated performance. For example, if the neural network is trained for a few epochs, it can save computation but it also means we have little (noisy) training information and therefore increase the chances of mistaking the extrapolation. It is important to note that extrapolated performance is not always absolute but it can also be relative to other candidates such as by predicting a ranking.

A shortcoming of the HPO early discarding literature is the multi-objective ((1) predictive performance, (2) overall computation) optimization viewpoint that such techniques are trying to solve. Therefore experimental evaluations lack comparison to proper baselines and sometimes present over-optimistic results. For example, it is common to compare early discarding techniques with complete training discarding (Falkner et al., 2018) and, only rare works consider the baseline performance which minimizes computation by stopping the training after a single epoch (Égelé et al., 2023b; Bohdal et al., 2023) during HPO and possibly selecting from the top-k models after further training. We call this baseline "1-Epoch" or more generally i-Epoch when the training is stopped after epoch i.

In this work, we evaluate the computation optimal policy 1-Epoch and show its surprising effectiveness in detecting top-ranked hyperparameter configurations. In addition, we look at the set of trade-offs between computation and predictive performance offered by different early discarding methods among which is the i-Epoch baseline. We do this

by spanning different levels of early discarding aggressiveness of each technique. Being more aggressive (*i.e.*, stopping training earlier) reduces computation but also generally sacrifices predictive performance. Therefore, we evaluate the multi-objective optimal frontier, also known as the Pareto-front, achieved by the different early discarding techniques. Ideally, varying the aggressiveness parameters of the different techniques, leads to a large Pareto-front, offering different trade-offs between aggressiveness (training epochs used) and predictive performance.

To simplify our experiments and avoid confounding factors, we do not use advanced HPO solvers but instead perform a random sampling of hyperparameter configurations, for which we can compare several early discarding techniques. We compare i-Epoch to asynchronous successive halving (SHA), parametric learning curve extrapolation (LCE), and the recently introduced LC-PFN model (Adriaensen et al., 2023) for learning curve extrapolation. We study these techniques on various classification and regression tasks for the class of fully connected deep neural networks. Against all expectations, our **principal findings** are:

1. dynamically allocating resources as done by successive halving or learning curve extrapolation offers minimal (and oftentimes no) utility compared to a constant number of training epochs, and

2. one can often early discard models after only one epoch without losing significant final predictive performance, indicating that perhaps learning curves are more well-behaved than one may expect.

We believe these findings highlight the necessity to incorporate 1-Epoch in future studies since it achieves such good predictive performance for minimal computation while being extremely simple to implement.

5.2 . Methods for vertical early discarding

We consider a function $f(\theta, i) \in \mathbb{R}$ that returns (empirical) generalization error of a deep neural network pipeline configured with hyperparameters $\theta \in \Theta$ (*i.e.*, a vector of mixed variables) after $i \in \mathcal{I}$ training epochs. In our setting we bound the number of training epochs $i_{\text{min}} \leq i \leq i_{\text{max}}$. Next consider a hyperparameter optimization algorithm $a \in \mathcal{A}$ such that $a(f, \Theta, \mathcal{I}) = (c_L, c_I)^T$ where $c_L = f(\theta^*, i_{\text{max}}) \in \mathbb{R}$ is the generalization error of the returned trained deep neural network pipeline configured with hyperparameters θ^* and $c_I \in \mathbb{N}$ is the total number of training epochs used by a to complete the hyperparameter optimization process. Then, the multi-objective problem that hyperparameter optimization with early discarding algorithms aims to solve is:

$$\begin{aligned} &\min_{a \in \mathcal{A}} \ (c_L, c_I) \\ &\text{s.t.} \quad (c_L, c_I)^T = a(f, \Theta, \mathcal{I}) \end{aligned} \tag{5.1}$$

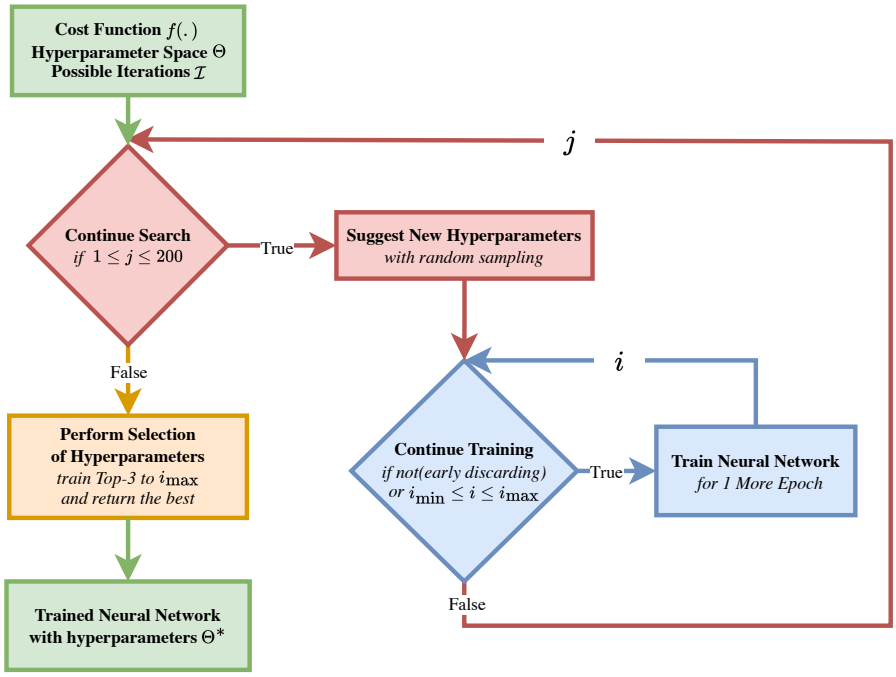


Figure 5.1: Hyperparameter optimization and its components including input/output, outer optimization loop exploring new hyperparameter configurations, inner optimization loop incrementally allocating training iterations (**what we study in this work**) and selection of hyperparameters to return. In *italic* we specify the blocks to match with our experimental study.

In Figure 5.1 we provide a flowchart diagram of the hyperparameter optimization with early discarding algorithm class $\mathcal{A}$ that we consider. The HPO process comprises an outer open cycle (red parts), in which an optimizer decides whether optimization should be continued or not. If so, it picks a candidate hyperparameter (HP) configuration (or various if parallelization is supported) for evaluation. Then, the performance of the chosen configurations is computed (blue parts). Since we only consider training of neural networks one can think of the candidate evaluation as an inner cycle in which an empirical learning curve is constructed, with one entry per epoch. In the orange box, a set of final candidates is selected (possibly of size 1) and trained to convergence (if not done already). Among these, the candidate with the best performance is returned and serves as a trained model for predictions.

In the interest of separation of concerns, this paper focuses only on the aspect of early discarding (blue diamond). The other components are fixed as follows: The outer cycle simulates a random search with an evaluation limit of 200 pipelines, which are sampled offline to make sure that all early discarding methods decide upon the same pipelines. Since no model of the performance landscape is built in the random search, the evaluation module simply returns the prediction performance of the network at the time when training is being stopped (no matter whether prematurely or because it has converged).

The orange component selects the 3 best configurations found during optimization and trains them to convergence (if not yet). It then returns the best of these models.

This being said, our study focuses only on early discarding techniques for *single candidates* as opposed to *candidate portfolios*. Many popular optimizers consider entire portfolios of candidates, which are then reduced at some predefined ratio (Jamieson and Talwalkar, 2016; Li et al., 2018; Falkner et al., 2018; Awad et al., 2021). We are interested in a more flexible class of early discarding techniques that do not need to know all the candidates upfront but decide only upon one candidate at a time based on the score of the best candidate seen so far. This is also referred to as the difference between horizontal optimization (simultaneously growing learning curves of a portfolio) and vertical optimization (evaluating candidates one by one, possibly without even knowing the whole set of candidates to be evaluated) (Mohr and van Rijn, 2022).

Among these early discarding techniques for single candidates, we consider three state-of-the-art approaches from different research branches and an approach that simply trains the networks for a previously defined constant number of epochs. First, for the idea of Successive Halving, which is used in many horizontal optimizers (Jamieson and Talwalkar, 2016; Li et al., 2018; Falkner et al., 2018; Awad et al., 2021), there is a sequential variant (Li et al., 2020) that can be used as an independent early discarding module. The second and third approaches discard candidates based on extrapolated learning curves using Monte Carlo Markov Chains (MCMC) (Égelé et al., 2023b) and Prior Fitted Networks (PFN) (Adriaensen et al., 2023), respectively. Another approach for extrapolation, learning curve-based cross-validation (LCCV) (Mohr and van Rijn, 2023) with state-of-the-art results in early discarding is not considered in the evaluation, because it is based on the assumption of convexity (or concavity) of the learning curves, which is the typical case for sample-wise learning curves but not iteration-wise learning curves as created during the training of a neural network (Mohr and van Rijn, 2022).

5.2.1 . Successive halving in the vertical setting (r-SHA)

Successive Halving (SHA) (Jamieson and Talwalkar, 2016) is an *optimization* technique that receives a *set* of candidates, which is successively reduced while granting more resources to candidates that are being retained. A common approach is to eliminate 50% of the candidates and double the amount of resources for the remaining candidates until only one candidate remains; thereby, all iterations consume roughly the same quantity of compute resources.

It is possible to isolate the idea of SHA to use it as an early discarding module such as shown in blue in Figure 5.1 (Li et al., 2020). To do this, one can test at epoch i if the currently observed score is among the top-$100/r\ \%$ already observed in the past for other candidates at the same epoch i, where r is called the reduction factor (*e.g.*, $r = 2$ for a reduction of 50%). If this is the case, then the training is continued otherwise it is stopped. This condition is not checked at every training epoch but follows a geometric schedule.

5.2.2 . Parametric learning curve extrapolation via MCMC (ρ-LCE)

Learning Curve Extrapolation (LCE) (Domhan et al., 2015) uses a parametric model

to predict the continuation of a learning curve. The parametric functions used for this task are mostly power laws originating from physics research (Mohr and van Rijn, 2022). It is also common to consider linear combinations of such functions (Domhan et al., 2015).

To enable a configurable greediness, we are interested in *probabilistic extrapolations*. That is, the extrapolation technique should output a *distribution* over learning curves rather than just a single one (usually the likelihood maximizer). These distributions can be obtained by sampling from the posterior distribution, usually using a Bayesian approach (Domhan et al., 2015; Klein et al., 2016).

However, we found that the above techniques suffer from instabilities, which is why we here use a technique called RoBER (Robust Bayesian Early Rejection) (Égelé et al., 2023b). Instead of considering a linear combination of several parametric models, we only consider one, that is MMF4 which was found to work well in general for extrapolation by (Mohr et al., 2022). In addition, for robustness, we use an approach that combines frequentist and Bayesian modeling. That is, first, we fit the parametric model using Levenberg-Marquadt, which minimizes the mean squared error on the observed anchors of the learning curve. Afterward, we use these fitting parameters $\hat{\nu} \in \mathbb{R}^{d_\nu}$ to derive a data-driven prior of the form $\nu \sim N(\hat{\nu}, 1)$. We use a Gaussian likelihood on the observed learning curve anchors with an exponential prior with scale parameter 1. This completely defines the posterior, which is sampled using Markov-Chain-Monte-Carlo (MCMC). This allows us to sample the distribution of extrapolated values at the largest anchor. We compute this distribution for each currently observed learning curve. If this distribution indicates for a learning curve candidate that we are with probability larger than ρ worse than the current best-observed learning curve value, the candidate is eliminated. The larger ρ, the more conservative: for example if $\rho = 0.9$, a candidate is only discarded if the probability that it under-performs the currently best one at the horizon is greater or equal to 90%.

5.2.3 . Learning curve extrapolation via prior fitted networks (ρ-PFN)

Prior Fitted Networks (PFN) are transformer networks that are being trained on synthetical tasks sampled from a so-called prior distribution (Müller et al., 2021). Here, tasks are described as labeled datasets together with unlabeled test points. For a new task, the PFN does not only output a single prediction for each test point but a *distribution*.

Due to their general nature, PFNs can also be used to predict distributions over learning curves. A recent approach that reports results comparable to or better than the MCMC approach of (Domhan et al., 2015) while being much faster was presented in (Adriaensen et al., 2023). In this approach, synthetic learning curves are sampled from a prior over linear combinations of model classes; a subset of those suggested in (Domhan et al., 2015) is used. The authors of this network offer a pre-trained implementation[1], which comes with an API that allows extrapolations of learning curves out of the box. Our experiments are based on this implementation.

As for LCE, one can define a confidence level ρ and discard candidates only if the prob-

[1] https://github.com/automl/lcpfn

ability that the limit performance is worse than the currently known best solution is at least ρ.

5.2.4 . Discarding after a constant number of training epochs (i-Epoch)

The last and simplest method is the one of a constant number of epochs. In this case, the number of epochs is defined a priori and does not depend on any observations made during the evaluation of the candidate. In our experiments, we consider all number of epochs between 1 and 100 as possible limits.

This method is different from the others in that it does not necessarily train *any* model to convergence *during* the evaluation. In all the other approaches, at least one network, namely the one that is believed to be best, is trained until convergence. On the contrary, in the case of a constant number of epochs, even the best network is not (necessarily) trained to convergence during evaluation but only in the final selection phase (orange box in Fig. 5.1). Of course, if the number of epochs configured is high, it can *implicitly* happen that the networks converge during evaluation. In particular, no early *stopping* (mind the difference to early discarding) is used to stop training once the curve has flattened out, so training can even take more epochs than what would be observed with a standard early stopping approach.

5.3 . Experimental design to compare early discarding strategies

Our experiments were designed to answer the following research questions (RQs) for the hyperparameter optimization of deep neural networks:

RQ1: What is the anytime performance of the HPO process (*i.e.*, when stopped at any iteration of the red loop in Figure 5.1) when run with the different early discarding techniques for extreme configurations of discarding aggressiveness (*i.e.*, when stopping training at the earliest and the latest)?

RQ2: For each early discarding technique, what is its Pareto-frontier in terms of (1) final predictive performance (of the selected and trained hyperparameter configuration) and (2) total training epochs consumed in the HPO process, obtained when testing different settings of the method?

RQ3: What does each method contribute to the Pareto-frontier resulting from all techniques? This aims to see if methods complement each other in terms of attainable trade-offs and which algorithm offers the most diverse trade-offs.

RQ4: How does 1-Epoch compare to other methods and how can we understand its surprisingly good performance?

Preempting the detailed results, we already summarize at this point that the answers to these questions might be in contrast to the expectations in two ways:

1. While it is clear that 1-Epoch is Pareto-optimal (since one cannot be faster), one would expect that i-Epoch tends to develop a sub-optimal Pareto frontier (compared to other early discarding methods) as i grows. This is because, since i-Epoch does not react to the previous performance observations, there is an increasing risk that (unpromising) candidates are trained unnecessarily long so that the number of total training epochs in the HPO increases without generating any benefit. In other words, for pretty much any $i > i_{\text{min}}$ for some small i_{min}, *e.g.*, 5 or 10, one would expect that there are configurations of the other early discarding methods that Pareto-dominate i-Epoch. The surprising insight of our experiments is that **the simple i-Epoch policy is rarely ever Pareto-dominated by any other method**.

2. While one would generally expect the maximally aggressive strategy 1-Epoch to deliver significantly sub-optimal results in predictive performance c_L, we show that generally **there is little and sometimes *no* possible improvement in predictive performance c_L over the 1-Epoch baseline**. In several cases, **1-Epoch is not only Pareto-optimal but strictly optimal**.

5.3.1 . Benchmarks of precomputed neural networks learning curves

To be able to generalize conclusions from this work, we answer the questions on several datasets, both regression and classification, which displayed noticeable differences in the observed learning curves. However, we limited our study to the class of fully connected deep neural networks, still including a variety of hyperparameters (*e.g.*, preprocessing, residual connections, regularization).

All learning curves used to benchmark early discarding techniques were computed and stored *prior* to the experimentation. We now describe this generating process. All evaluated deep neural networks are trained for 100 epochs, which fixes $i_{\text{min}} = 1$ and $i_{\text{max}} = 100$. For *regression* tasks, we used an external benchmark of pre-computed learning curves from HPOBench (Eggensperger et al., 2021a; Klein and Hutter, 2019a). The deep neural networks from this benchmark are similar to ours but were generated from 9 hyperparameters listed in Table 5.1 and 4 datasets were used.

Hyperparameters	**Choices**
Initial LR	$\{0.0005, 0.001, 0.005, 0.01, 0.05, 0.1\}$
Batch Size	$\{8, 16, 32, 64\}$
LR Schedule	$\{\text{cosine}, \text{fix}\}$
Activation/Layer 1	$\{\text{relu}, \text{tanh}\}$
Activation/Layer 2	$\{\text{relu}, \text{tanh}\}$
Layer 1 Size	$\{16, 32, 64, 128, 256, 512\}$
Layer 2 Size	$\{16, 32, 64, 128, 256, 512\}$
Dropout/Layer 1	$\{0.0, 0.3, 0.6\}$
Dropout/Layer 2	$\{0.0, 0.3, 0.6\}$

Table 5.1: Hyperparameter search space for regression benchmarks defined in HPOBench (Eggensperger et al., 2021a; Klein and Hutter, 2019a).

Datasets were split into 3 folds. The training split was used to optimize the neural network weights for a fixed hyperparameter configuration. The validation split was used to optimize the hyperparameter configurations and serves as an estimate of generalization performance. The test split was used as a final set of data to provide an unbiased report of our results. The data split was 60% for training, 20% for validation, and 20% for testing in the regression tasks, which was dictated by the setup of (Eggensperger et al., 2021a). In the classification tasks, we chose the split to be 80% for training, 10% for validation, and 10% for test.

For classification tasks, we generated a set of 1,000 randomly sampled hyperparameter configurations from a search space of 17 hyperparameters listed in Table 5.2. The learning curve generation for each classification task required about 1 hour of computing on 400 parallel NVIDIA A100 GPUs on the Polaris Supercomputer at the Argonne Leadership Computing Facility.

Hyperparameters	**Choices**
Activation Function	{none, relu, sigmoid, softmax, softplus, softsign, tanh, selu, elu, exponential}
Activity Regularizer	{none, L1, L2, L1L2}
Batch Normalization	{True, False}
Batch Size	$[1, 512]$ (log-scale)
Bias Regularizer	{none, L1, L2, L1L2}
Dropout Rate	$[0.0, 0.9]$
Kernel Initializer	{random-normal, random-uniform, truncated-normal, zeros, ones, glorot-normal, glorot-uniform, he-normal, he-uniform, orthogonal, variance-scaling}
Kernel Regularizer	{none, L1, L2, L1L2}
Learning Rate	$[10^{-5}, 10^{1}]$ (log-scale)
Number of Layers	$[1, 20]$
Number of Units	$[1, 200]$ (log-scale)
Optimizer	{SGD, RMSprop, Adam, Adadelta, Adagrad, Adamax, Nadam, Ftrl}
Regularizer Factor	$[0.0, 1.0]$
Shuffle Each Epoch	{True, False}
Skip Connections	{True, False}
Transform Categorical	{onehot, ordinal}
Transform Real	{minmax, std, none}

Table 5.2: Hyperparameter search space for classification benchmarks.

For all these configurations we compute the training, validation, and test learning curves by collecting confusion matrices on predictions. Accounting for hyperparameter configurations that resulted in failures (*e.g.*, "`nan`" loss with overflow or underflow) we end up with about 850 correct learning curves for each classification dataset. The diversity of evaluated tasks is provided through the number of samples, features, classes or targets, and the type of features (real or categorical) in Table 5.3.

5.3.2 . Experimental protocol

Dataset (OpenML-Id)	#Features	#Samples	#Classes or #Targets	Real Features	Categorical Features
Slice Localization (42973)	380	53,500	1	True	False
Protein Structure (44963)	9	45,730	1	True	False
Naval Propulsion (44969)	14	11,934	1	True	False
Parkinson's Telemonitoring (4531)	20	5,875	2	True	True
MNIST (554)	784	70,000	10	True	False
Australian Electricity Market (151)	8	45,312	2	True	True
Bank Marketing (1461)	16	45,211	2	True	True
Letter Recognition (6)	16	20,000	26	True	False
Letter Speech Recognition (300)	617	7,797	26	True	False
Robot Navigation (1497)	24	5,456	4	True	False
Chess End-Game (3)	36	3,196	2	False	True
Multiple Features (Karhunen) (14)	76	2,000	10	True	False
Multiple Features (Fourier) (16)	64	2,000	10	True	False
Steel Plates Faults (40982)	27	1,941	7	True	False
QSAR Biodegradation (1494)	41	1,055	2	True	False
German Credit (31)	20	1,000	2	True	True
Blood Transfusion (1464)	4	748	2	True	False

Table 5.3: Characteristics of datasets used for our experiments. On Top, the 4 datasets were used for regression, and on the bottom, the 10 datasets were used for classification. The datasets are sorted by decreasing the number of samples.

As we are interested in evaluating early discarding techniques (blue diamond in Figure 5.1) isolated from the process which suggests hyperparameter configurations, we propose the following experimental protocol. The simulated process that suggests hyperparameter configurations (red rectangle in Figure 5.1) is a random sampling from the set of pre-computed learning curves. This process is fixed by an initial random seed to simulate the same stream of candidate learning curves to different early discarding techniques. The number of search iterations (red diamond in Figure 5.1) is fixed to 200 (main constant which makes outcomes of all experiments comparable). Once the (red) loop of 200 candidates is over, the Top-3 models observed are selected and trained to completion if not already done. A model that was not trained to completion during the previous 200 iterations will be retrained from scratch. Of course, these additional training epochs are accounted for in the total number of training epochs used by the method. For example, in 1-Epoch after 200 iterations we select the Top-3 candidates based on the observed scores c_L, we train them to completion so it consumes an additional 3×100 epochs, then we return the best from these 3. For 100-Epoch, as all evaluated models are already trained to completion no additional training is required. The performance we report corresponds to the score reached by "*Method + Top-3*" at any iteration of the search. This corresponds to looking at the "*any-time*" performance of each early discarding method, that is looking at what would be the outcome of the method if we were to stop after k hyperparameter search iterations (red loop) for all $k \in [i_{\text{min}} = 1, i_{\text{max}} = 100]$. A fixed set of 10 random seeds is set to perform 10 repetitions for each method. This protocol ensured that each method was exposed to the same streams of candidates. Therefore the different out-

comes observed are only coming from differences in the decisions taken by each method to stop or continue the training.

5.3.3 . Performance indicators

In this section, we describe the two performance indicators of importance in our study. First, we detail the R^2 metric (generalized to both regression and classification) used to assess the predictive performance of evaluated hyperparameter configurations. Then, we detail the hypervolume indicator (HVI) metric used to assess the quality of the solutions for multi-objective optimization.

First, we introduce the coefficient of determination R^2 in the case of regression tasks, where the target prediction is a real value, and, then we extend the notion to the case of classification tasks, where the target prediction is a categorical value in the spirit of (El-Yaniv et al., 2017), also called the Prediction Advantage. This metric is useful as it standardizes both regression and classification similarly which helps us homogenize regression and classification learning curves. A dataset $D = \{(x_1, y_1), ..., (x_n, y_n)\}$ is composed of i.i.d. variables from the joint distribution $P(X, Y)$. In *regression*, the usual definition of R^2 (*a.k.a.*, coefficient of determination) is:

$$R^2 := 1 - \frac{SS_{res}}{SS_{tot}} = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}(x_i))^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{5.2}$$

where SS_{res} is the residual sum of squares, SS_{tot} is the total sum of squares, $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$ is the empirical mean of the marginal distribution $P(Y)$ and, $\hat{y}(x_i)$ is a prediction from our model. This definition can also written as:

$$R^2 = 1 - \frac{\frac{1}{n}\sum_{i=1}^{n} L_2(y_i, \hat{y}(x_i))}{\frac{1}{n}\sum_{i=1}^{n} L_2(y_i, \bar{y})} \approx 1 - \frac{E[(Y - E[Y|X])^2|X]}{E[(Y - E[Y])^2]} \tag{5.3}$$

In the form given by Equation 5.3, the expectations $E[Y]$ and $E[Y|X]$ correspond to the optimal Bayes predictors for the squared loss $L_2(Y, \hat{Y}) = (Y - \hat{Y})^2$ respectively on the marginal and conditional distributions. Therefore R^2 corresponds to the normalization of the expected error of the optimal Bayes predictor on the conditional $P(Y|X)$ distribution by the expected error of the optimal Bayes predictor on the marginal distribution $P(Y)$ (*a.k.a.*, constant or "dummy" predictor). In *classification*, we replace the squared-loss with the $0-1$ loss $L_{0\text{-}1}(Y, \hat{Y}) = 1$ if $Y \neq \hat{Y}$ else 0. The optimal Bayes predictor becomes the mode instead of the mean (*i.e.*, class with the highest probability). We then obtain a new definition of R^2 for classification:

$$R^2 = 1 - \frac{\frac{1}{n}\sum_{i} L_{0\text{-}1}(y_i, \hat{y}(x_i))}{\frac{1}{n}\sum_{i} L_{0\text{-}1}(y_i, \dot{y})} \tag{5.4}$$

where $\dot{y}$ is the mode on the marginal distribution $P(Y)$. This is also known as the Prediction Advantage (El-Yaniv et al., 2017). For both regression and classification, we have that performance of zero means that the model is as bad as the optimal constant predictor that only uses information from the marginal $P(Y)$ and ignores the input X. If the R^2 is

1 the prediction is "*perfect*" (which also means that there is no presence of random noise on the target). In our study, **the goal is to maximize the R^2 score for improved predictive performance** which is equivalent to minimizing $c_L := 1 - R^2(\theta, i_{\text{max}})$ in Equation 5.1 ($f(\theta, i)$ returns the $1 - R^2$ score for the hyperparameters θ at training epoch i).

Now that we have discussed the performance indicator for prediction we will present the metric used to assess the quality of multi-objective optimization (MOO). For the sake of brevity, we will not recall the formal definitions related to the notion of Pareto-optimality in MOO. However, shortly we recall that Pareto-Front refers to the solution set in the objective space (*i.e.*, 2-dimensional in our case as we have 2 objectives c_L and c_I). As these objectives are (supposedly) conflicting, c_L the predictive performance and c_I the total number of training iterations used, the Pareto-Front is a one-dimensional space (*i.e.*, a line) unless the problem is "degenerated", meaning there is no real conflict between objectives and the solution set is therefore containing a single point. Among the possible metrics used in MOO (Audet et al., 2021) and as we do not know the true Pareto-Front of our problem we decide to use the hypervolume indicator (HVI). As we are in 2-D it corresponds to measuring the area defined by an estimated Pareto-Front and a reference point (fixed for all experiments on the same dataset). The HVI is compliant with the notion of Pareto-optimality and also known to measure the compare the diversity of solutions (*i.e.*, trade-offs) between different Pareto-Fronts. In our study, **the goal is to identify the early discarding technique which maximizes the Hypervolume indicator when evaluated at different levels of aggressiveness**.

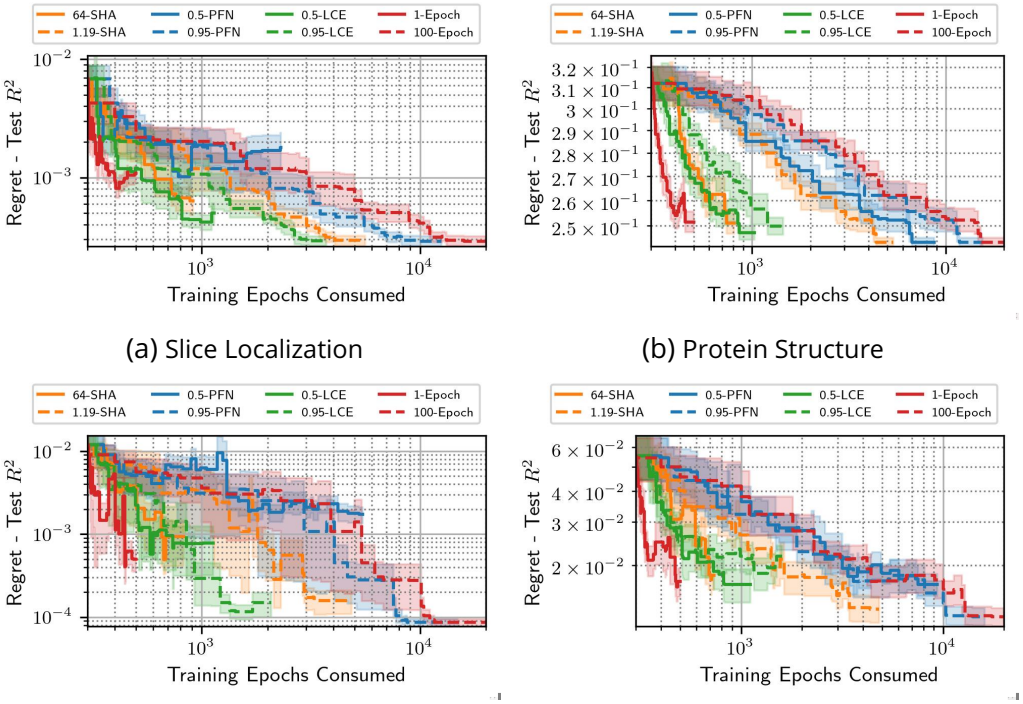
5.4 . Experimental results

In this section, we present the results that helped us answer the research questions introduced in Section 5.3.

5.4.1 . Anytime performance of early discarding techniques (RQ1)

To understand the anytime performance of early discarding techniques we plot the $1 - R^2$ *test* performance as a function of overall training epochs realized so far. That is, a curve that passes the point (t, l) in the plot means that the test score of the model that *would* have been picked if the HPO process had stopped after t total training epochs would have been l. This type of performance curve weighs training epochs equally for all hyperparameter configurations, which may be deceiving since they can vary in computational cost (*e.g.*, large and small neural networks). Still, it is a convenient simple method abstracting from implementation details. We present the performance curves in Figures 5.2 and 5.3 for classification and regression respectively.

The most important insight from the plots is that the sensitivity of the early discarding techniques with respect to their aggressiveness parameter varies a lot. While the i-Epoch and r-SHA algorithms are very sensitive to the aggressiveness (as expected), the learning curve extrapolation-based methods (*i.e.*, ρ-LCE and ρ-PFN) are surprisingly less sensitive to aggressiveness parameter ρ. This can be observed especially on the set of classification tasks shown in Figure 5.3. In other words, for ρ-LCE and ρ-PFN, it almost makes no differ-



(a) Slice Localization

(b) Protein Structure

(c) Naval Propulsion

(d) Parkinson's Telemonitoring

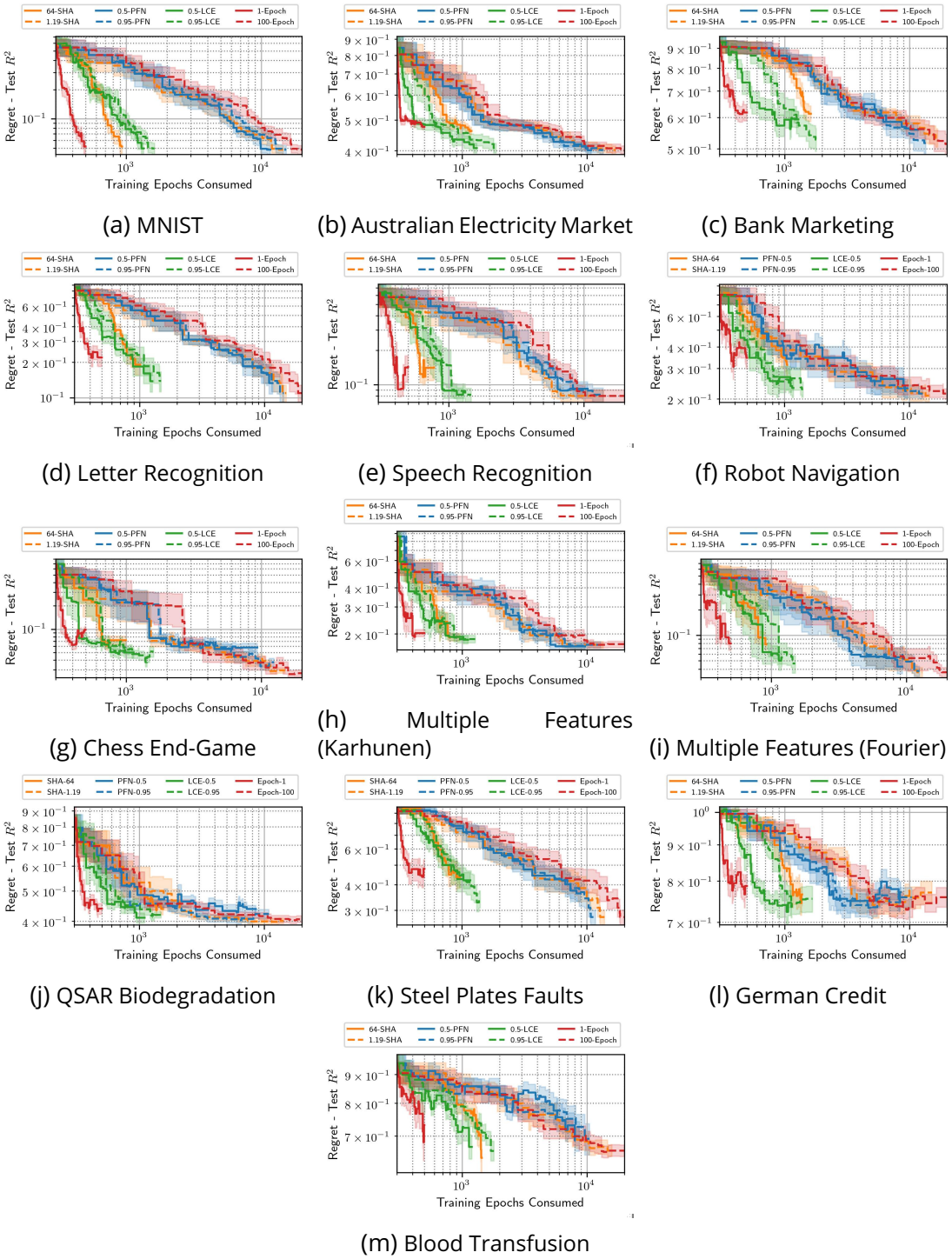
Figure 5.2: Comparing the any-time performance of various early discarding techniques during a random search (mean and one standard error over 10 repetitions) of 200 iterations (4 regression tasks). The two baseline strategies 1-Epoch and 100-Epoch method bound the trade-offs that can be achieved. **The predictive performance of 1-Epoch is at least of the same order of magnitude as other strategies while consuming a significantly smaller (the minimum in training epochs) number of training epochs**.

ence in consumed training epochs whether the user requires almost certainty ($\rho = 0.95$) or whether the certainty is just as good as a coin flip ($\rho = 0.5$). This could indicate that the models express too little uncertainty about the extrapolated learning curve.

Another observation is that the ρ-PFN method hardly reduces the overall training epochs used by 100-Epoch as can be seen for all datasets. It means that the learning curve extrapolation of this method is probably over-optimistic. It even seems to perform worse than 100-Epoch for both predictive performance and overall training epochs used on learning curves which are very noisy and increasing. These failures can be observed in Figures 5.3c, 5.3l and 5.3m.

A third observation is that ρ-LCE while being a more robust version of LCE, can still under-perform in achieving predictive performance even when being set to be conservative (ρ=0.95). This can be seen in Figures 5.2d, 5.3d, 5.3h and 5.3k. This confirms our belief that such models express too little uncertainty about the extrapolation.

From the practical viewpoint, no utopia method has yet been found. A utopia method would achieve a strict and consistent dominance compared to the 100-Epoch baseline.



(a) MNIST

(b) Australian Electricity Market

(c) Bank Marketing

(d) Letter Recognition

(e) Speech Recognition

(f) Robot Navigation

(g) Chess End-Game

(h) Multiple Features (Karhunen)

(i) Multiple Features (Fourier)

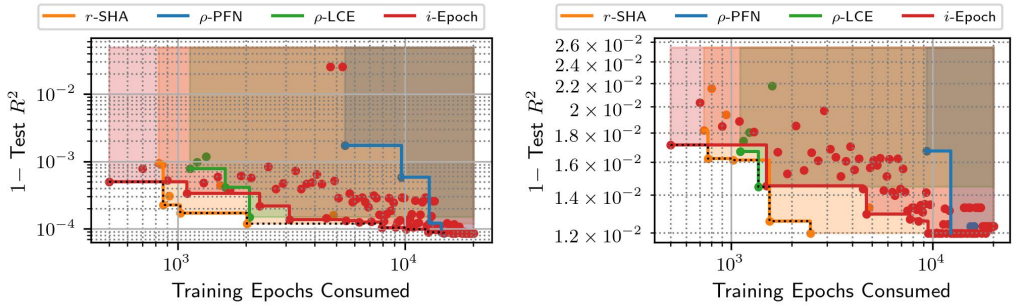
(j) QSAR Biodegradation

(k) Steel Plates Faults

(l) German Credit

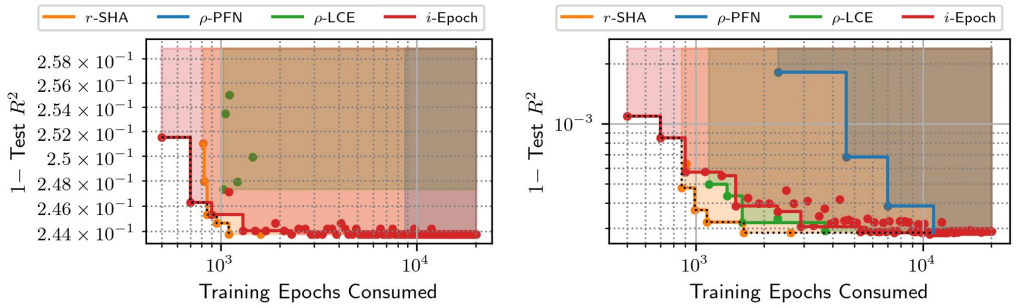
(m) Blood Transfusion

Figure 5.3: Comparing the any-time performance of various early discarding techniques during a random search (mean and one standard error over 10 repetitions) of 200 iterations (on 13 classification tasks).



(a) Naval Propulsion

(b) Parkinson's Telemonitoring



(c) Protein Structure

(d) Slice Localization

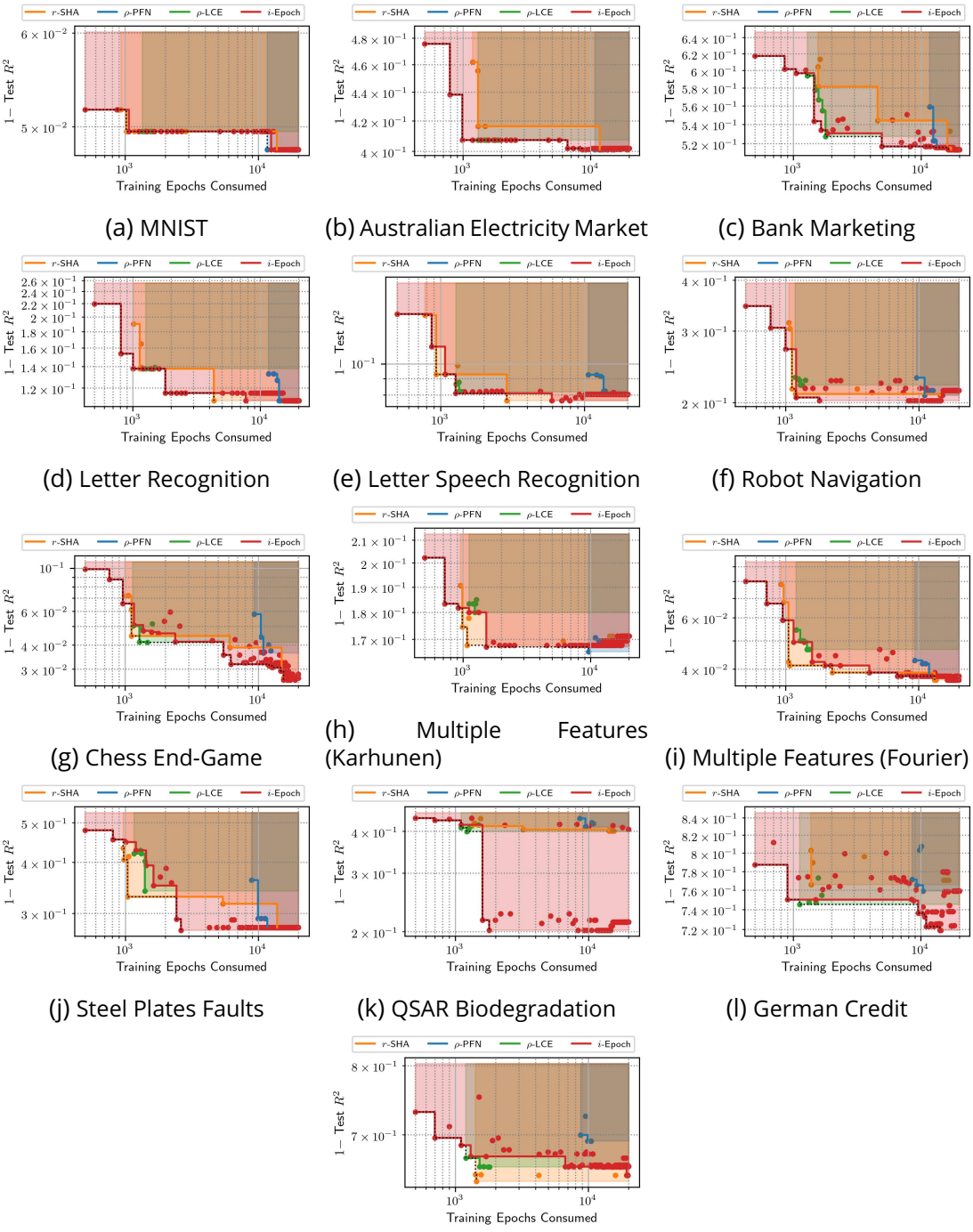
Figure 5.4: Multi-objective profiles built from spanning various levels of aggressiveness of early discarding methods (on 4 regression tasks). The estimated Pareto-Front of each method is shown in a plane line. The black dotted line corresponds to the estimated Pareto-Front including the methods altogether. **It can be seen that the i-Epoch strategy spans more trade-offs (larger area) than other methods while never being significantly dominated**.

That is a method that achieves, on all tasks, better predictive performance while being faster than the base full training evaluation. Such a method seems not to exist currently and may not exist if both objectives c_L and c_I are truly conflicting.

Finally, the presented performance curve plots also show the importance of considering the 1-Epoch baseline to contextualize results and avoid an overly optimistic presentation of the methods. Without the solid red line which corresponds to 1-Epoch, ρ-LCE might appear a quite dominant approach in this experimental setting. While it is true that learning curve extrapolation-based methods are very convincing in many cases, there are several datasets, such as Protein Structure (Figure 5.2b), Parkinson's Telemonitoring (Figure 5.2d), MNIST (Figure 5.3a), QSAR-Biodegradation (Figure 5.3j), or German Credit (Figure 5.3l), in which the 1-Epoch baseline can reduce the number of epochs of LCE again by about 50% without losing significant or any predictive performance.

5.4.2 . Trade-offs between predictive accuracy and speed (RQ2)

While the previous question only considers two extreme configurations to understand the sensitivity of the HPO process concerning the aggressiveness of the early discarding



(a) MNIST

(b) Australian Electricity Market

(c) Bank Marketing

(d) Letter Recognition

(e) Letter Speech Recognition

(f) Robot Navigation

(g) Chess End-Game

(h) Multiple Features (Karhunen)

(i) Multiple Features (Fourier)

(j) Steel Plates Faults

(k) QSAR Biodegradation

(l) German Credit

(m) Blood Transfusion

Figure 5.5: Multi-objective profiles built from spanning various levels of aggressiveness of early discarding methods (13 classification tasks). The estimated Pareto-Front of each method is shown in a plane line. The black dotted line corresponds to the estimated Pareto-Front including the methods altogether. **It can be seen that the i-Epoch strategy spans more trade-offs (larger area) than other methods while never being significantly dominated**.

technique, we now want to better understand the actual trade-offs that each method can span. At this point, we no longer look at any time performance but instead, we look at the final predictive performance and overall consumed training epochs for one aggressiveness setting. Once all methods and all aggressiveness levels are collected we compute the Pareto-Front of each early discarding method which does not always contain all evaluated points.

From the results presented in the previous section, we already know that the Pareto-Fronts of ρ-PFN will be strictly dominated by other techniques (*i.e.*, the area/hypervolume it defines will be strictly included in the area of other methods). Since even the difference between minimum ($\rho = 0.95$) and maximum aggressiveness ($\rho = 0.5$) had only minimal effect, one expects the area covered by ρ-PFN in the multi-objective profile to be narrow.

The multi-objective profiles and the corresponding Pareto-Fronts are presented in Figures 5.4 and 5.5. For i-Epoch, a value was computed for each $1 \leq i \leq 100$. For ρ-LCE and ρ-PFN, we used values of $\rho \in \{0.5, 0.7, 0.8, 0.9, 0.95\}$, and for r-SHA we used values of $r \in \{\sqrt{\sqrt{2}} = 1.19, \sqrt{2} = 1.41, 2, 4, 8, 16, 32, 64\}$. For each approach, the Pareto-optimal points are connected with a step function to indicate the respective Pareto frontier. The shaded areas show the hypervolume of each approach.

Some plots, like in Fig. 5.4a, suggest a certain inconsistency in the trade-off logic of i-Epoch in the sense that many points of a single method do not lie on the same method's Pareto frontier. However, this can often be attributed to noise on rather small scales. For example, in the mentioned plot, differences are on a scale below 10^{-3}, *i.e.*, less than 0.1% difference in performance in terms of the constant predictor baseline. For the other methods, this effect is less pronounced or does not occur because much fewer points are generated and the change in aggressiveness is more significant (10%-steps in the case of ρ compared to single epochs in the case of i-Epoch).

The first observation confirms our expectation that learning curve extrapolation-based techniques offer little diversity of trade-offs. ρ-LCE, no matter how aggressiveness is configured, tends to use about 10x less training epoch than 100-Epoch while sometimes slightly under-performing in attained predictive performance. And, again, PFN on most datasets offers almost no reductions regardless of the configuration of ρ.

5.4.3 . Identifying the most complete early discarding technique (RQ3)

To quantity the observation that i-Epoch offers a more diverse set of trade-offs we compute the relative hypervolume spanned by each method in Figures 5.4 and 5.5. To evaluate the hypervolume we set as reference point $c_{\text{ref}} := (\max \mu_L + \sigma_L^{\text{err}}, \max \mu_B + \sigma_B^{\text{err}})$ (*i.e.*, element-wise upper-bound of observations) for all methods. Then we apply a $\log_{10}(.)$ transformations on both c_L and c_I values (including the reference point). This transformation serves to spread the volume contributed by small and large values equally. Otherwise, differences in hypervolume would become unnoticeable as soon as improvements in c_L or c_I become the order of magnitude smaller than the largest reference point values. Finally, we compute the hypervolume of all methods which we divide the hypervolume of the Pareto-Front considering all observations (in dotted black line). This relative hypervolume then quantifies how much each method contributes to the available set of trade-offs

that we observed. The closer is the value to 1 the more complete the method. The resulting scores are presented in Table 5.4.

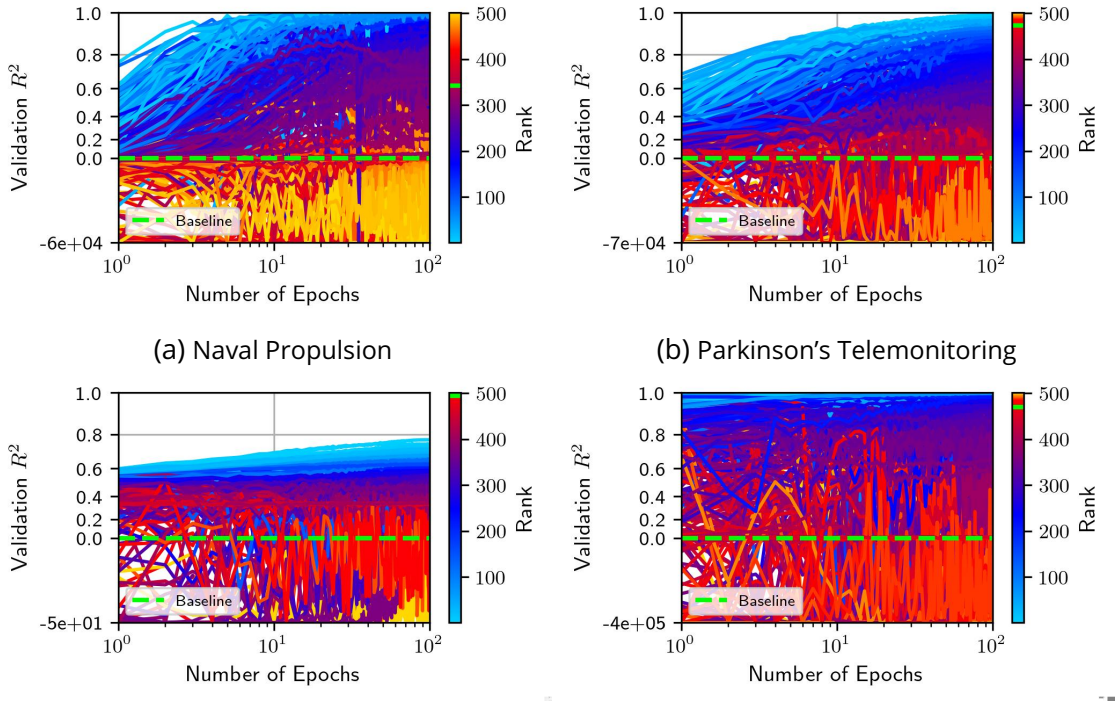
As it can be observed i-Epoch achieves the highest scores on all but one task giving it an average rank of 1.125. The second best-ranked method is ρ-SHA followed by ρ-LCE. The ρ-PFN method consistently finishes last ranked on all tested tasks. Lastly, we also notice that **relative hypervolume scores of i-Epoch are often close to 1 which confirms that this method spans most of the observed trade-offs and it is never significantly outperformed in either objective**.

Dataset	r-SHA	ρ-PFN	ρ-LCE	i-Epoch
Slice Localization	0.930	0.401	0.823	**0.932**
Protein Structure	0.916	0.241	0.652	**0.989**
Naval Propulsion	0.881	0.280	0.742	**0.951**
Parkinson's Telemonitoring	**0.930**	0.201	0.667	0.880
MNIST	0.858	0.176	0.743	**0.994**
Australian Electricity Market	0.768	0.205	0.829	**1.000**
Bank Marketing	0.609	0.184	0.847	**0.989**
Letter Recognition	0.851	0.169	0.672	**0.988**
Letter Speech Recognition	0.915	0.175	0.810	**0.974**
Robot Navigation	0.882	0.218	0.789	**0.992**
Chess End-Game	0.866	0.233	0.827	**0.965**
Multiple Features (Karhunen)	0.901	0.231	0.606	**0.955**
Multiple Features (Fourier)	0.936	0.239	0.697	**0.951**
Steel Plates Faults	0.806	0.257	0.644	**0.923**
QSAR Biodegradation	0.141	0.037	0.176	**0.993**
German Credit	0.585	0.198	0.800	**0.970**
Blood Transfusion	0.811	0.167	0.753	**0.856**
Average Rank	2.029	4.000	2.846	**1.125**

Table 5.4: Relative hypervolumes of each early discarding technique concerning the hypervolume including all the techniques. Bold and green is best, followed by yellow, orange and red. These scores assess the diversity of trade-offs, in consumed training epochs and predictive performance, offered by each technique among all observed outcomes. The higher the score the more complete (in terms of possible trade-offs) is the early discarding technique. In our experiments, **the i-Epoch technique offers the best set of trade-offs** and achieves a trade-off close to 1 indicating optimality amongst all methods.

5.4.4 . The unreasonable effectiveness of early discarding after 1-Epoch (RQ4)

Last but not least, throughout our presented results we can notice the unreasonable effectiveness of 1-Epoch. Despite sometimes being noisier in its performance profiles such as in Figures 5.2c, 5.5f and,5.5l, it always achieved better any-time performance than other early discarding methods. This is demonstrated by the fact that its performance curve does not cross with the performance curves of other methods. However, the difference in final predictive performance c_L can sometimes be statistically significant such as in



(a) Naval Propulsion

(b) Parkinson's Telemonitoring

(c) Protein Structure

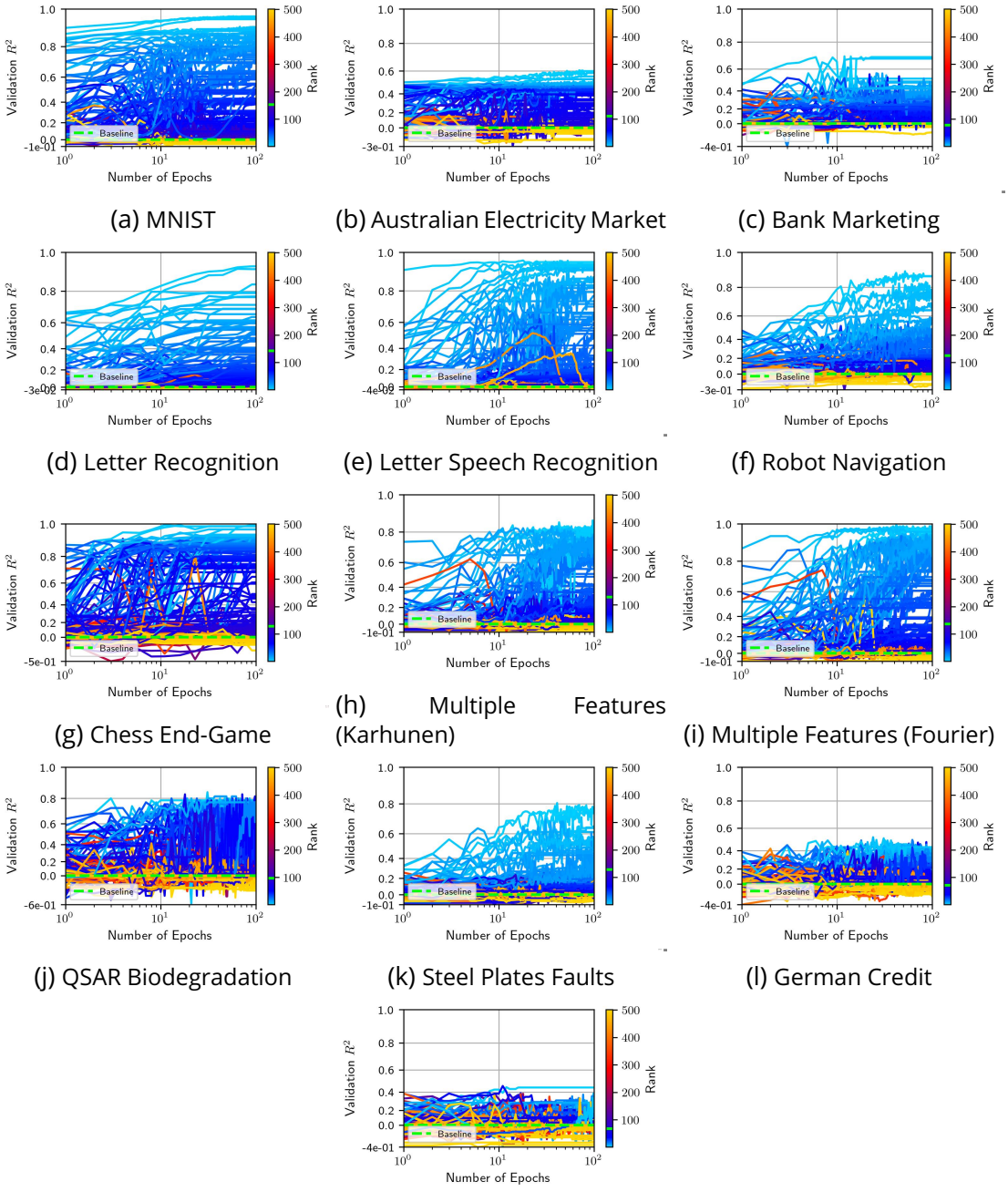
(d) Slice Localization

Figure 5.6: Visualizing the final ranking for good (light blue) and bad (yellow) models for 500 randomly sampled learning curves (on 4 regression tasks). The constant predictor performance (at 0) is shown as a green dashed line. **Models can be selected from the first epoch as there appear to be dominant models early on in the training epochs**.

Figures 5.3c, 5.3g, 5.3i and, 5.3k which confirms the trade-off between the two objectives.

How is it possible this approach can perform so well? To better understand this, we analyze the learning curves of our experiments. In Figures 5.6 and 5.7 we display 500 randomly sampled learning curves from our pre-computed sets, we then color the curves by their ranking at 100 epochs (the maximum number of training epochs). Low ranks, colored in light blue, correspond to the best models, while high ranks, colored in red and then yellow, correspond to the worst models. We plot the performance of the constant predictor as a dashed lime line and also plot its rank.

In these plots, it can be observed that **for all benchmarks there exists among the best models some that are also the best early in the training process**. This observation explains the performance of 1-Epoch. Then, in a few cases, we can observe **a significant proportion of models perform worse than the constant predictor**. It is about 33% of models in Figure 5.6a and about 80% of models in 5.7g to 5.7i. Finally, it seems that **learning curve oscillations are correlated with the final predictive performance**. The best models present much fewer oscillations than the worst models, which justifies high aggressiveness in the early discarding method.



(a) MNIST (b) Australian Electricity Market (c) Bank Marketing

(d) Letter Recognition (e) Letter Speech Recognition (f) Robot Navigation

(g) Chess End-Game (h) Multiple Features (Karhunen) (i) Multiple Features (Fourier)

(j) QSAR Biodegradation (k) Steel Plates Faults (l) German Credit

(m) Blood Transfusion

Figure 5.7: Visualizing the final ranking for good (light blue) and bad (yellow) models for 500 randomly sampled learning curves (on 13 classification tasks). The constant predictor performance (at 0) is shown as a green dashed line. **Models can be selected from the first epoch as there appear to be dominant models early on in the training epochs**.

5.5 . Conclusion

In this paper, we conducted a comprehensive analysis of early discarding techniques for hyperparameter optimization of fully connected deep neural networks. Our study rigorously compared an array of advanced techniques and unveiled intriguing findings: (1) **the unreasonable effectiveness of the 1-Epoch strategy**, a straightforward yet previously overlooked baseline method, and (2) **the Pareto-dominance of the i-Epoch strategy despite its simplicity**.

We attribute the success of this strategy to effectively differentiating between high and low-potential models in the early stages of training. Notably, models with promising prospects exhibit minimal performance oscillations, a pattern consistently observed in widely used benchmarks. These insights not only underscore the importance of incorporating the i-Epoch strategy in future benchmark analyses but also highlight the potential necessity of considering the multi-objective problem hidden behind early discarding strategies. **An early discarding method would bring significant value only if it complements or dominates the i-Epoch Pareto-Front**. Current early discarding approaches only add minimal or no utility in this sense.

Besides its good performance, we believe that 1-Epoch's simplicity is valuable in itself. Besides being easy to implement, before execution, it is easy to predict the number of training epochs consumed by i-Epoch for any i when it is not possible for either ρ-LCE or r-SHA. This makes i-Epoch practically attractive.

To be noted, our work is limited to using "epoch" as iteration units for early discarding. While this is convenient and appealing to conduct studies independent of hardware implementation considerations, practical application settings may require considering wall time or other options as units for early discarding. In particular, since different configurations may have different batch sizes, some configurations could be much faster to train than others. However, comparing wall-clock time is extremely hardware and software implementation dependent. Maybe considering the size of the deep neural network as a third objective of Equation 5.1 could be an improvement.

We have tried a limited range for the aggressiveness parameters of ρ-LCE and r-SHA. Their Pareto-Front could be larger and more dominant for a wider range of parameters considered. However, values of $\rho < 0.5$ seem relatively strange for ρ-LCE because in that case it will be very pessimistic about extrapolated performance and discard models as soon as there is a small probability of under-performing. r-SHA could be more aggressive but it should be noted that our largest reduction factor of 64 corresponds to continuing training only if the model is in the current Top-1.5% meaning comparing to the single best model after 100 Hyperparameter suggestions and Top-3 of 200. Also, this value is significantly larger than the suggested default value of 4 in the original paper(Li et al., 2020).

Also, we studied the early discarding methods in combination with a random search. In other words, HPO is often combined with techniques that suggest candidates use more sophisticated methods, such as Bayesian optimization or portfolio (Jamieson and Talwalkar, 2016; Li et al., 2017; Falkner et al., 2018; Awad et al., 2021). However, for such ap-

proaches we cannot quantify the computational cost as easily through the number of epochs, as the Bayesian optimization may not be a neural net. Besides that, the comparison becomes more complicated, because the different components (configuration proposer, early discarding technique, etc.) may interact in unexpected ways. Therefore, such a comparison is out of the scope.

To come back to the question of the earlier work: is one epoch all you need? We think the answer remains to be seen, in particular, we think that the 1-epoch approach can be even pushed further. During the first epoch, more information is available for making decisions. For example, the loss per batch could be collected. This again forms a curve of performances versus the number of batches processed, which seems conceptually similar to a learning curve. This curve could be extrapolated as well. This will allow us to make potentially better decisions after 1 epoch or even training could be stopped before finishing one epoch. The latter could be especially promising for large language models, for which one epoch of training can already consume hours of training time.

5.6 . Acknowledgment

This research was conducted in collaboration with Felix Mohr, Tom Viering, Venkatram Vishwanath, and Prasanna Balaprakash. We would also like to thank Isabelle Guyon for participating with us in discussions that improved the quality of this work. This material is based upon work supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. This research used resources from the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility. This material is based upon work supported by ANR Chair of Artificial Intelligence HUMANIA ANR-19-CHIA-0022 and TAILOR EU Horizon 2020 grant 952215. Felix Mohr participated through the project ING-312-2023 from Universidad de La Sabana, Campus del Puente del Común, Km. 7, Autopista Norte de Bogotá. Chía, Cundinamarca, Colombia.

6 - Uncertainty quantification through deep ensembles from hyperparameter optimization checkpoints

Through previous chapters we understood how to efficiently explore configurations of learning workflows. However, the resulting predictors generally did not estimate uncertainty. In the case of deep neural networks, ensembles can be used to quantify such uncertainty and hard know to be competitive with Bayesian approaches while benefiting from better computational scalability. Still, building a good ensemble of deep neural networks is hard because the hyperparameters of each member of this ensemble needs to be optimized for improved diversity (known quality of a good ensemble). To address this issue, in this chapter we propose AutoDEUQ, an automated approach for generating an ensemble of deep neural networks. Our approach leverages the joint neural architecture and hyperparameter search to generate ensembles. We use the law of total variance to decompose the predictive variance of deep ensembles into aleatoric (data) and epistemic (model) uncertainties. We show that AutoDEUQ outperforms state-of-the-hart methods such as probabilistic backpropagation, Monte Carlo dropout, deep ensemble, distribution-free ensembles, and hyper ensemble methods on a number of regression benchmarks.

6.1 . Introduction to deep neural network uncertainty quantification

Uncertainty quantification (UQ) for machine-learning-based predictive models is crucial for assessing the trustworthiness of predictions from the trained model. For deep neural networks (DNNs), it is desirable for predictions to be accompanied with estimates of uncertainty because of the black-box nature of the function approximation. Two major forms of uncertainty exist (Hüllermeier and Waegeman, 2021): aleatoric data uncertainty and epistemic model uncertainty. The former occurs due to the inherent variability or noise in the data. The latter is attributed to the uncertainty associated with the neural network model parameter estimation or out-of-distribution predictions. The epistemic uncertainty increases in the regions that are not well represented in the training dataset (Gal and Ghahramani, 2016a). While the aleatoric uncertainty is irreducible, the epistemic uncertainty can be reduced by collecting more training data in the appropriate regions.

Several researchers have looked at extending deterministic neural networks to probabilistic models. A strongly advocated method is to have a fully Bayesian formulation, where each trainable parameter in a Dneural network is assumed to be sampled from a very high-dimensional (and arbitrary) joint distribution (Neal, 2012). However, this is computationally infeasible, for example because of issues of convergence, for any practical deep learning tasks with millions of trainable parameters in the architecture and having large datasets. Consequently, several approximations to fully Bayesian formulations have been put forth to reduce the computational complexity of uncertainty quantification in DNNs. These range from simple augmentations such as the mean-field approximation in Bayesian backpropagation via variational inference (Hoffman et al., 2013; Hernández-Lobato and Adams, 2015), where each parameter is assumed to be sampled from an independent unimodal Gaussian distribution, to Monte Carlo dropout (Srivastava et al., 2014), where random neurons are switched off during training and inference to obtain ensemble predictions.

Ensemble methods that utilize multiple independently trained DNNs have shown considerable promise for uncertainty quantification (Lakshminarayanan et al., 2017; Ovadia et al., 2019; Ashukha et al., 2020) by outperforming conventional approximations to the fully Bayesian methodology. Blundell et al. (Wilson and Izmailov, 2020) argue that the deep ensembles approach is fully congruous with Bayesian model averaging, which attempts to estimate the posterior distribution of the targets given input data by marginalizing the parameters. However, a key factor in deep ensembles is model diversity without which uncertainty cannot be captured efficiently. For example, in (Lakshminarayanan et al., 2017), each member of the ensemble has an identical neural architecture and is trained by using maximum likelihood or maximum a posteriori optimization through different initialization of weights. Consequently, ensemble diversity is limited since each model can at best settle on distinct local minima. Marginalization over these models in the ensemble will force the function approximation to collapse on one hypothesis and provide results similar to

Bayesian model averaging for a single architecture with probabilistic trainable parameters. Such an implicit assumption may be undesirable when dealing with datasets that are generated from a combination of hypotheses. Moreover, the lack of flexibility in the ensemble may lead to a poorer estimate of epistemic uncertainty. Although Wenzel et al. (Wenzel et al., 2021) attempted to relax this issue by allowing more diversity in the ensemble, they vary just two hyperparameters. Similarly, Zaidi et al. (Zaidi et al., 2020) vary the architecture with fixed trainable hyperparameters to increase the ensemble diversity. By constructing diverse DNNs models through a methodical and automated approach, we hypothesize that the assumption of and the eventual collapse to one hypothesis can be avoided, thus providing robust and efficient estimates of uncertainty.

6.1.1 . Overview of problems and methods

To model aleatoric uncertainty, one must model the conditional distribution $\mathrm{p}\left(y \mid x\right)$ for the target y given an input x. One way is to assume that this distribution is Gaussian and then estimate its parameters (mean and variance) (Nix and Weigend, 1994). However, these estimates summarize conditional distributions into scalar values and are thus unable to model more complex profiles of uncertainty such as multimodal or heteroscedastic profiles (*i.e.*, where the noise depends on the input x). To resolve this issue, one can use implicit generative models (Mohamed and Lakshminarayanan, 2016) and mixture density networks (Bishop, 1994). A different approach is deep kernel learning (van Amersfoort et al., 2021), which extracts kernels and uses them in Gaussian-process-based methods for datasets with large features and sample size. However, this adds additional complexity because one must find the correct hyperparameters. An alternative strategy is to directly output prediction intervals from the neural network, such as in (Pearce et al., 2018), which has the advantage of not requiring any distribution assumption on the output variables. However, these methods are ill-equipped to quantify epistemic uncertainty.

Several methods for epistemic uncertainty have been proposed. Bayesian neural networks (Bneural networks) (Maddox et al., 2019) and deep ensembles (Caruana et al., 2006) are the main approaches. In Bneural network, the weights are assumed to follow a joint distribution, and the epistemic uncertainty is quantified through Bayesian inference. Except for trivial cases, however, Bayesian inference is computationally intractable. Therefore, several approximations to Bneural network have been proposed, such as probabilistic backpropagation (PBP) (Hernández-Lobato and Adams, 2015) and Bayes by Backprop (Blundell et al., 2015). In deep ensembles (Lakshminarayanan et al., 2017), multiple networks are aggregated to quantify the uncertainty. Each network in the ensemble provides an estimate of aleatoric uncertainty, while their aggregation provides an estimate of epistemic uncertainty. However, the members of such ensembles often have similar architecture and hyperparameter values but with different weights generated through random weight initialization in addition to the stochastic aspect of the training procedure. Recently, new automated methods were proposed to improve deep ensembles, wherein hyperparameters (Wenzel et al., 2021) or neural architecture decision variables (Zaidi et al., 2020) are varied to improve the diversity of models in the ensemble to achieve improved aleatoric and epistemic uncertainty estimates.

Recently, Russell and Reale (Russell and Reale, 2021) developed a joint covariance matrix with end-to-end training using a Kalman filter to represent aleatoric uncertainty while using dropout to estimate the epistemic component. Although not an ensemble method, it models aleatoric and epistemic at the same time.

6.1.2 . Contributions

Given training and validation data, the proposed AutoDEUQ method (i) starts from a user-defined neural architecture and hyperparameter search space; (ii) leverages aging evolution and Bayesian optimization methods to automatically tune the architecture decision variables and training hyperparameters, respectively; (iii) builds a catalog of models from the search; and (iv) uses a greedy heuristic to select models from the catalog to construct ensembles. The predictions from the ensemble models are then used to estimate the aleatoric and epistemic uncertainty. AutoDEUQ is built on the successes of three recent works in the deep ensemble literature: deep ensemble (Lakshminarayanan et al., 2017), hyper ensemble (Wenzel et al., 2021), and neural ensemble search (Zaidi et al., 2020). However, our AutoDEUQ method differs from deep ensemble in the following ways. While aleatoric and epistemic uncertainties are modeled empirically, we theoretically decompose the predicted variance of deep ensembles into its aleatoric and epistemic components. Moreover, in AutoDEUQ, the Dneural network architectures and the training hyperparameter values in the ensembles are different, and more importantly they are generated automatically. While hyper ensemble and neural ensemble methods explore hyperparameters and architectural choices, respectively, and generate ensembles, AutoDEUQ explores both spaces simultaneously. The key contributions of the paper are as follows: (1) automation of deep ensembles construction with joint neural architecture and hyperparameter search and (2) demonstration of improved uncertainty quantification compared with prior ensemble methods and, consequently, advancement of state of the art in deep ensembles.

6.2 . Automated deep ensemble with uncertainty quantification (AutoDEUQ)

We focus on uncertainty estimation in a regression setting. Our methodology, automated deep ensemble for uncertainty quantification (AutoDEUQ), estimates aleatoric and epistemic uncertainties by automatically generating a catalog of neural networks through joint neural architecture and hyperparameter search, wherein each model is trained to minimize the negative log likelihood to capture aleatoric uncertainty, and selecting a set of models from the catalog to construct the ensembles and model epistemic uncertainty without losing the quality of aleatoric uncertainty.

6.2.1 . Uncertainty quantification in deep ensembles

In supervised learning, the dataset D is composed of n realisations (x_i, y_i) from *i.i.d.* random variables with support $\mathcal{X} \times \mathcal{Y}$, where x_i are called the inputs and y_i are called the targets. Here, we focus on regression problems, wherein the targets are real valued. Given D, we seek to approximate the true probability density function $\mathrm{p}\left(y|x\right)$ using an ensemble

of trained predictors α, in our case deep neural networks. Each neural network is used as a model $\mathrm{p}_\alpha(y|x)$ of the true density function $\mathrm{p}(y|x)$. Following previous work (Lakshminarayanan et al., 2017), we assume that the true distribution follows a normal law $\mathrm{P}(Y|X) \sim \mathcal{N}(\mu, \sigma^2)$ with mean μ and variance σ^2. Then, a neural network is model of the density of this law, $\mathrm{p}_\alpha(y|x) \sim \mathcal{N}(\mu_\alpha(x), \sigma^2_\alpha(x))$ where $\mu_\alpha(x)$ is the learned predicted mean and, $\sigma^2_\alpha(x)$ is the learned predicted variance. The later being an approximation of the true conditional variance $\mathbb{V}_Y[Y|X=x]$, also known as aleatoric uncertainty. The epistemic uncertainty, related to the model this time, is estimated through the ensemble of deep neural networks $\mathrm{p}_\xi(y|x)$

We represent each neural network α_{θ,w_θ} by a tuple of its hyperparameters $\theta \in \Theta$ and weights $w_\theta \in \mathcal{W}(\theta)$. For simplicity we will use the α notation in place of α_{θ,w_θ} in the remaning of the chapter. Where Θ is the hyperparameter space and $\mathcal{W}(\theta)$ is the space of weights that can be attained given hyperparameters θ. Hyperparameters are partitionned into $\theta = (\theta_a, \theta_p) \in \Theta_a \times \Theta_p$, where Θ_a represents the neural network architecture hyperparameters (*i.e.*, describing the neural network topology), Θ_p represents neural network training pipeline hyperparameters (*e.g.*, data preprocessing, learning rate, batch size).

As neural networks are trained to learn mean $\mu_\alpha(x)$ and variance $\sigma^2_\alpha(x)$, the goal is to find α such that the the likelihood $\mathrm{p}(D|\alpha)$ of the data D is maximised. As described by Lakshminarayanan et al. (2017), this corresponds to finding α such that the negative log-likelihood loss is minimized (as opposed to the usual squared error), given by:

$$L(x, y; \alpha) = -\log \mathrm{p}_\alpha(y|x) = -\frac{\log\left(2\pi\sigma^2_\alpha(x)\right)}{2} - \frac{\left(y - \mu_\alpha(x)\right)^2}{2\sigma^2_\alpha(x)} \tag{6.1}$$

To model epistemic uncertainty, we build an ensembles of such trained neural networks (Lakshminarayanan et al., 2017). In our approach, we first generate a catalog of $I \in \mathbb{Z}$ checkpointed neural networks $\mathbf{A} := \{\alpha_1, \ldots, \alpha_I\}$ (where α represents the neural network architecture, training pipeline hyperparameters, and weights) during a hyperparameter optimization. Then, we iteratively select $K \in \mathbb{Z}$ of these checkpointed models (with replacement) to form an ensemble $\xi := \{\alpha_1, \ldots, \alpha_K\}$. This ensemble then represents a mixture distribution with density $\mathrm{p}_\xi(y|x) = \frac{1}{K}\sum_{k=1}^{K} \mathrm{p}_{\alpha_i}(y|x)$ also approximating the true density $\mathrm{p}(y|x)$. The mean of the mixture density is $\mu_\xi(x) := \frac{1}{K}\sum_{k=1}^{K} \mu_\alpha(x)$, and the variance (Rudary, 2009) is given by applying the law of total variance (Theorem 3.1):

$$\sigma^2_\xi(x) = \underbrace{\mathbb{E}_A\left[\sigma^2_A(x)\right]}_{\text{Aleatoric Uncertainty}} + \underbrace{\mathbb{V}_A\left[\mu_A(x)\right]}_{\text{Epistemic Uncertainty}}, \tag{6.2}$$

where $\mathbb{E}[.]$ is the expectation, $\mathbb{V}[.]$ is the variance and A is a random variable corresponding to the deep neural network predictor. Equation (6.2) formally provides the decomposition of overall uncertainty of the ensemble into its individual components such that $\mathbb{E}_A\left[\sigma^2_A(x)\right]$ marginalizes the effect of A (the learned model) and captures the aleatoric uncertainty and $\mathbb{V}_A\left[\mu_A(x)\right]$ captures the variability of predictions between different models, therefore capturing the epistemic uncertainty (*a.k.a.*, uncertainty about the model).

We write the empirical estimate of the mean and variance of the ensemble as:

$$\begin{aligned}\mu_\xi(x) &= \frac{1}{K}\sum_{k=1}^{K}\mu_{\alpha_k}(x)\\ \sigma_\xi^2(x) &= \underbrace{\frac{1}{K}\sum_{k=1}^{K}\sigma_{\alpha_k}^2(x)}_{\text{Aleatoric Uncertainty}} + \underbrace{\frac{1}{K}\sum_{k=1}^{K}(\mu_{\alpha_k}(x)-\mu_\xi(x))^2}_{\text{Epistemic Uncertainty}},\end{aligned} \tag{6.3}$$

where K is the size of the ensemble. The total uncertainty quantified by $\sigma_\xi^2(x)$ is a combination of aleatoric and epistemic uncertainty, which are given by the the mean of the predictive variance of each model in the ensemble and the predictive variance of the mean of each model in the ensemble.

6.2.2 . Building an ensemble from a diverse catalog of deep neural networks

Let D be decomposed as $D = D_{\text{train}} \cup D_{\text{valid}} \cup D_{\text{test}}$, referring to the training, validation, and test data, respectively. A neural architecture configuration θ_a is a vector from the neural architecture search space Θ_a, defined by a set of neural architecture decision variables. A hyperparameter configuration θ_p is a vector from the training pipeline hyperparameter search space Θ_p defined by a set of hyperparameters used for training or preprocessing the data (*e.g.*, data normalization, learning rate, batch size). Following Problem 2.2, the empirical problem of joint neural architecture and hyperparameter search can be formulated as:

$$\begin{aligned}\theta^* &= \arg\min_{\theta\in\Theta}\frac{1}{|D_{\text{valid}}|}\sum_{(x_i,y_i)\in D_{\text{valid}}} L\left(x_i, y_i; \alpha_{\theta, w_{\theta^*}}\right)\\ \text{s.t. } {w_\theta}^* &= \arg\min_{w_\theta\in\mathcal{W}(\theta)}\frac{1}{|D_{\text{train}}|}\sum_{(x_j,y_j)\in D_{\text{train}}} L\left(x_j, y_j; \alpha_{\theta, w_\theta}\right)\end{aligned} \tag{6.4}$$

where $L\left(.\right)$ is the negative log-likelihood (Equation 6.1), x_i, y_i are samples from the dataset, the hyperparameters θ include both the neural architecture θ_a and the pipeline hyperparameters θ_p, and w_θ are weights of the neural network for the corresponding hyperparameters. The hyperparameter are selected based on the validation set D_{valid} and the weights are selected based on the training set D_{train}. As the sub-problem is solved through the learner (Definition 2.4) the empirical problem we solve can be written as:

$$\Theta^* = \arg\min_{\theta\in\Theta}\frac{1}{|D_{\text{valid}}|}\sum_{(x_i,y_i)\in D_{\text{valid}}} L\left(x_i, y_i; \beta_\theta(D_{\text{train}}, \mathcal{E})\right) \tag{6.5}$$

where Θ^* is a random variable because of the randomness from $\mathcal{E}$ the randomized learner (*e.g.*, , weight initialization, stochastic gradient descent). The pseudo code of the AutoDEUQ is shown in Algorithm 4. To perform a joint neural architecture and hyperparameter search, we leverage aging evolution with asynchronous Bayesian optimization (AgEBO) (Égelé et al., 2021).

Aging evolution (AgE) (Real et al., 2018) is a centralized parallel neural architecture search (NAS) method for searching over the neural architecture space. The AgEBO method

Algorithm 4: Automated Deep Ensemble with Uncertainty Quantification (AutoDEUQ)

Inputs : `thetaSpace`: a configuration space, `nInitial`: the number of initial hyperparameter configurations for Bayesian optimization, `f`: a function that returns the cost of the learning workflow, `nPopulation`: the population size for aging evolution, `nSample`: the sample size for aging evolution, `nWorker`: the number of parallel workers, `nEnsemble`: the number of unique elements in the ensemble.

Output: `ensemble` the ensemble of selected trained deep neural networks.

`thetaArray`, `costArray` ← New empty arrays of hyperparameter configurations and costs ;
`catalog` ← New empty list ;
`model` ← New surrogate model for Bayesian optimization ;
`population` ← New empty queue of size `nPopulation` for Aging Evolution;
`nFreeWorker` ← `nWorker` ;
`/* Hyperparameter optimization (with AgEBO (Égelé et al., 2021)) */`
while *stopping criteria not valid* **do**
  for $i \in [|1, nFreeWorker|]$ **do**
    `/* Select pipeline hyperparameters with (asynchronous) Bayesian optimiaztion */`
    if *Length of* $thetaArray < nInitial$ **then**
      `thetaPipeline` ← Sample pipeline hyperparameters from `thetaSpace` ;
    else
      Update `model` with pipeline hyperparameters from `thetaArray` and `costArray` ;
      `thetaPipeline` ← Returns `theta` in `thetaSpace` that minimizes the acquisition function ;
    end
    `/* Select neural architecture hyperparameters with (asynchronous) Aging Evolution */`
    if *Length of* $population < nPopulation$ **then**
      `thetaArchitecture` ← Sample neural architecture hyperparameters from `thetaSpace` ;
    else
      `sample` ← Select randomly without replacement `nSample` elements from `population` ;
      `parentArchitecture` ← Select the element from `sample` with lowest cost ;
      `thetaArchitecture` ← Apply a random mutation to `parentArchitecture` ;
    end
    `theta` ← Concatenate architecture and pipeline hyperparameters ;
    Submit `theta` to be evaluated by a free worker ;
    `nFreeWorker` ← `nFreeWorker` $- 1$;
  end
  `weightDone`, `thetaDone`, `costDone` ← Returns weights, hyperparameters, and costs of completed evaluations ;
  Push `thetaDone`, `costDone` in `population` queue ;
  `nFreeWorker` ← `nFreeWorker` $+ length($`thetaDone`$)$;
  `thetaArray`, `costArray` ← Concatenate `thetaArray` with `thetaDone` and `costArray` with `costDone`;
  `catalog` ← Add new trained neural networks (`thetaDone`, `weightDone`) to `catalog` ;
end
`/* Ensemble Construction (with greedy selection (Caruana et al., 2004)) */`
`ensemble` ← New empty list ;
`ensembleLoss` ← $+\infty$
while *Unique elements in* $ensemble \leq nEnsemble$ **do**
  `modelLoss`, `model` ← Returns loss and `model` in `catalog` for which `ensemble` $\cup$ {`model`} has minimal loss on the validation dataset ;
  if $modelLoss \leq ensembleLoss$ **then**
    `ensemble` ← `ensemble` $\cup$ {`model`} ;
    `ensembleLoss` ← `modelLoss` ;
  else
    return `ensemble`
  end
end

follows the manager-worker paradigm, wherein a manager process runs a search method to generate multiple neural networks (including architecture and pipeline hyperparameters). Several parallel worker processes train these neural networks simultaneously The AgEBO method starts by sampling `nFreeWorker` neural architecture pipeline hyperparameters (lines 9 and 15). The neural networks obtained by using these concatenated hyperparameter (line 21) are sent for simultaneous evaluation on `nFreeWorker` workers (line 22). Then, we check whether any of the workers finish their evaluation (line 25), collect trained weights, hyperparameters and validation metrics. These results are used to generate the next set of neural architectures (lines 11-12) and pipeline hyperparameters (lines 17-19) to fill up free workers. For select the next neural architecture, from the incumbent `population`, `nSample` neural networks are sampled (line 17). The best neural architecture is selected from this `sample` (line 18). A random mutation is applied to generate a child architecture configuration (line 19). This mutation is obtained by first randomly selecting an architecture decision variable from the selected neural network and replacing its value with another randomly selected value excluding the current value. The AgEBO optimizes the pipeline hyperparameters by using a centralized asynchronous Bayesian optimization (asynchronous centralized and parallel version of Algorithm 1). To generate multiple multiple pipeline hyperparameters at the same time, the Bayesian optimization leverages a multipoint acquisition function based on a constant liar strategy (Ginsbourger et al., 2010a).

The catalog `catalog` of neural networks is obtained by storing all the evaluated neural networks (line 29). Then, to build the ensemble `ensemble`, we adopt a greedy selection strategy (lines 31-41) (Caruana et al., 2004). At each step, the neural network from the catalog that most improves the negative log likelihood (Equation 6.1 on the validation data D_{valid} of the incumbent ensemble is added to the ensemble (lines 34-36). The greedy approach can work well when the validation data is representative of the generalisation task (*i.e.*, big enough, diverse enough, with good coverage) (Caruana et al., 2004).

6.3 . Experimental results

We first describe the search space used in AutoDEUQ. Next, using a one-dimensional dataset, we present an ablation study to analyze the impact of different components of AutoDEUQ. Then, we compare AutoDEUQ with other methods.

6.3.1 . Defining a search space of deep neural network learning workflows

The neural architecture search space is modeled by using a directed acyclic graph , which starts and ends with input and output nodes, respectively (Figure 6.1). They represent the input and output layer of neural network, respectively. Between the two are intermediate nodes defined by a series of variable $\mathcal{N}$ and skip connection $\mathcal{SC}$ nodes. Both types of nodes correspond to categorical decision variables. The variable nodes model dense layers with a list of different layer configurations. The skip connection node creates a skip connection between the variable nodes. This second type of node can take two values: disable or create the skip connection. For a given pair of consecutive variable nodes

$\mathcal{N}_k$, $\mathcal{N}_{k+1}$, three skip connection nodes $\mathcal{SC}_{k-3}^{k+1}, \mathcal{SC}_{k-2}^{k+1}, \mathcal{SC}_{k-1}^{k+1}$ are created. These nodes allow for connection to the previous nonconsecutive variable nodes $\mathcal{N}_{k-3}, \mathcal{N}_{k-2}, \mathcal{N}_{k-1}$, respectively.

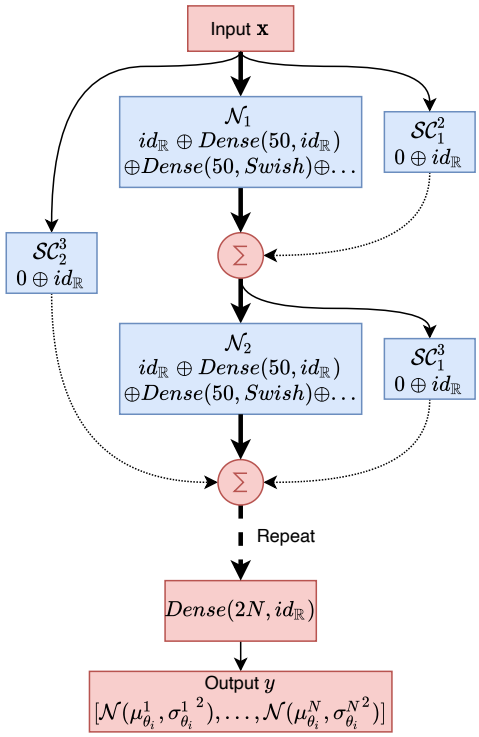


Figure 6.1: Search space of fully connected neural networks with regression outputs.

Each dense layer configuration is defined by the number of units and the activation function. We used values in $\{16, 32, \ldots 256\}$ and {elu, gelu, hard sigmoid, linear (*i.e.*, identity), relu, selu, sigmoid, softplus, softsign, swish, and tanh}, respectively. These resulted in 177 (16 units $\times$ 11 activation functions, and identity) dense layer types for each variable node. Skip connections can be created from at most 3 previous dense layers. Each skip connection is created with a linear projection so that feature vectors match in shape, and then addition is used to merge the vectors. The number of variable nodes is set to 3 for the one-dimensional toy dataset and to 5 for the regression benchmarks.

For the pipeline hyperparameter search space, we use a learning rate in the continuous range $[10^{-4}, 10^{-1}]$ with a log-uniform prior; a batch size in the discrete range $[1, 2, 3, \ldots, b_{max}]$ (where $b_{max} = 32$ for the toy example and $b_{max} = 256$ for the benchmark) with a log-uniform prior; an optimizer in { sgd, rmsprop, adagrad, adam, adadelta, adamax, nadam }; a patience number for the reduction of the learning rate in the discrete range $[10, 11, \ldots, 20]$, and a patience number for early stopping in the discrete range $[20, 21, \ldots, 30]$. The neural networks are trained with 200 epochs for the toy example and 100 epochs for the benchmark. The search space is the same for the toy and the benchmark. Models are checkpointed during their evaluation based on the minimum validation loss achieved. Input and output variables are standardized to have a mean of zero and a unit variance.

The experiments were conducted on the ThetaGPU system at the Argonne Leadership Computing Facility. ThetaGPU is composed of 24 nodes, each composed of 8 NVIDIA A100 GPUs and 2 AMD Rome 64-core CPUs. For the **generation of a catalog of models** we use different allocations (*i.e.*, number of nodes) depending on the dataset size. During the search, 1 process only using the CPU is allocated for the search algorithm; then neural network configurations (hyperparameters and architecture) are sent to parallel workers for the training. Each worker corresponds to a single GPU. Therefore, 1 node had 8 parallel

workers. For the **construction of an ensemble**, we load all checkpointed models on different GPU instances to perform parallel inferences and then save the predictions to apply the greedy strategy. On the software side, we used Python 3.8.5. The core of our dependencies is composed of TensorFlow 2.5.0, TensorFlow-Probability 0.13.0, Ray 1.4.0, Scikit-Learn 0.24.2, and Scipy 1.7.0.

6.3.2 . Qualitative assessement on a toy example

We follow the ideas from (Hernández-Lobato and Adams, 2015) to assess qualitatively the effectiveness of AutoDEUQ on a one-dimensional dataset. However, instead of the unimodal dataset generated from the cubic function used in (Hernández-Lobato and Adams, 2015), we used the $y = 2\sin(x) + \epsilon(x)$ sine function. We generated 200 points randomly sampled from a uniform prior in the x-range $[-30, -20]$ with $\epsilon \sim \mathcal{N}(0, 0.25)$ and 200 other points randomly sampled in the x-range $[20, 30]$ with $\epsilon \sim \mathcal{N}(0, 1)$. These 400 points constitute $D_{\text{train}} \cup D_{\text{valid}}$. We used random sampling to split the generated data: 2/3 for training and 1/3 for validation datasets. The two x-ranges are sampled with different noise levels to assess the learning of aleatoric uncertainty. The test set comprised 200 x-values regularly spaced between $[-40, 40]$, and the corresponding y values were given by $2\sin x$ with $\epsilon(x) = 0$. Consequently, we had three different ranges of x-values to assess epistemic uncertainty: training region, $[-30, -20]$ and $[20, 30]$; interpolation region, $[-20, 20]$; and extrapolation region: $[-40, -30]$ and $[30, 40]$. We seek to verify that the proposed method can model the aleatoric (different noise levels in the training region) and epistemic uncertainty (interpolation and extrapolation regions).

Importance of exploring diverse learning strategies and neural architectures

We perform an ablation study to show the effectiveness of tuning both architecture decision variables and training hyperparameters in AutoDEUQ. First, we designed a high-performing neural network by manually tuning the architecture decision variables and hyperparameter configurations on the validation data. We ran AutoDEUQ, which used AgEBO for catalog generation and the greedy model selection method for ensemble construction. Next, we used two AutoDEUQ variants: (1) AutoDEUQ (AgE), which used only AgE to explore the search space of the architecture space but used the hand-tuned hyperparameter values following the approach from (Zaidi et al., 2020), and (2) AutoDEUQ (BO), which used the hand-tuned neural architecture and used BO to tune the hyperparameters following the approach from (Wenzel et al., 2021). Finally, we switched off both AgE and BO and trained the manually generated baseline with 500 random-weight initializations to build the catalog. All these methods used greedy selection to build an ensemble of size $K = 5$ from their respective catalog of 500 models.

Figure 6.2 shows the results of these variants. We observe that the proposed AutoDEUQ (Figure 6.2.a) obtains a superior aleatoric and epistemic uncertainty estimation. The two different noise levels in the training region are well captured by the aleatoric uncertainty estimate. In the interpolation region, aleatoric uncertainty follows the noise levels of the nearby region. We also observe that epistemic uncertainty grows as we move

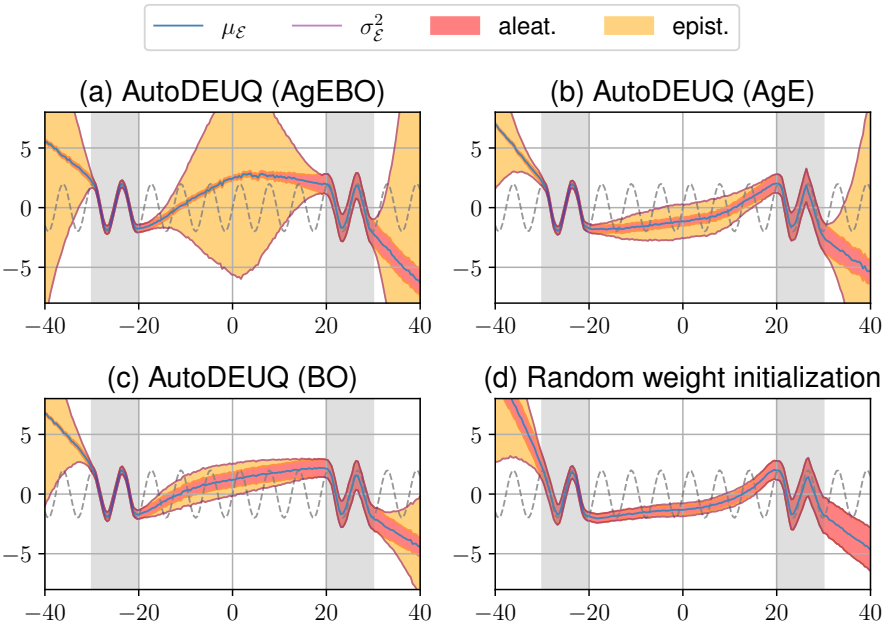


Figure 6.2: Ablation study of catalog generation: We progressively removed the different algorithmic components of AutoDEUQ and analyzed their impact on the uncertainty estimation.

from the training data region (grey). Moreover, we observe that its magnitude is large for the extrapolation region compared with the interpolation regions. Unlike AutoDEUQ (AgE) and AutoDEUQ (BO), the epistemic uncertainty grows from $x = -20$, peaks near $x = 0$, and becomes zero near $x = 20$. The results of AutoDEUQ (AgE) and AutoDEUQ (BO) variants are similar: while the aleatoric uncertainty estimates are good, both suffer from poor epistemic uncertainty estimation in the interpolation region. This can be attributed to a lack of model diversity in the ensemble, the former with fixed hyperparameters and the latter with fixed architectures. We observe that the random initialization strategy (Figure 6.2.d) with the hand-tuned neural architecture did not model epistemic uncertainty well. This result can be attributed to the simplicity of the dataset: given its low dimension, for the same architecture and hyperparameter configurations, the training results in similar neural networks.

Role of the hyperparameter optimization strategy

We analyze the impact of different search methods in AutoDEUQ on the uncertainty estimation. We compare the default AutoDEUQ (AgEBO) method (Figure 6.2.a) with random search (RS-Mixed) (Figure 6.3.a), AgE (AgE-Mixed) (Figure 6.3.b), and BO (BO-Mixed) (Figure 6.3.c). Note that RS, AgE, and BO do not consider the architecture and hyperparameter space separately. Instead, a configuration in the search space is given by a single vector of architecture decision variables and training hyperparameters.

We observe that the uncertainty estimates from the AutoDEUQ (RS-Mixed) are infe-

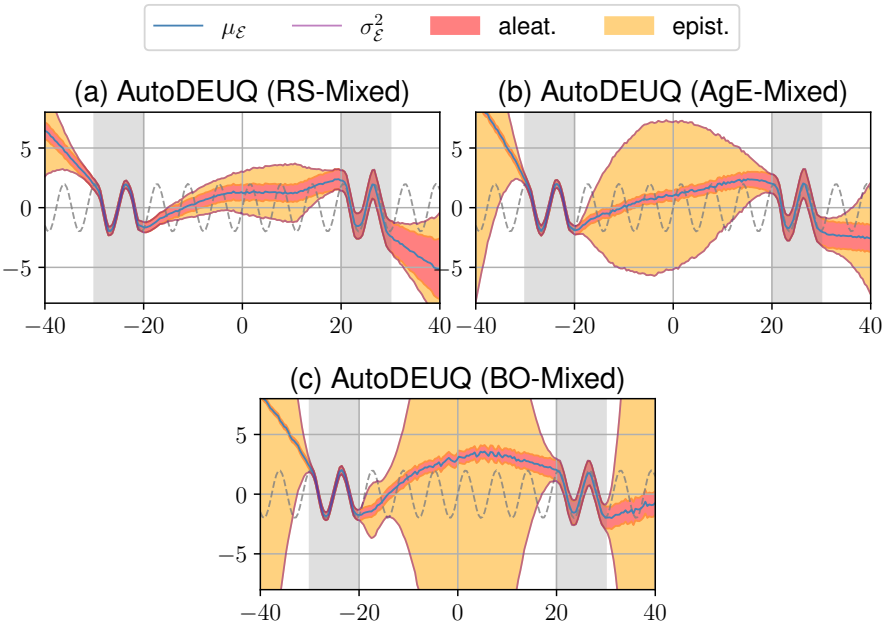


Figure 6.3: Comparison of different search methods in AutoDEUQ and their impact on uncertainty estimation.

Dataset	NLL						RMSE					
	PBP	MC Dropout	Deep Ens.	Hyper Ens.	DF Ens.	AutoDEUQ	PBP	MC Dropout	Deep Ens.	Hyper Ens.	DF Ens.	AutoDEUQ
boston	2.57	2.46	2.41	**2.15 (0.22)**	2.74	2.46 (0.09)	3.01	2.97	3.28	**2.87 (0.1)**	3.38	3.09 (0.31)
concrete	3.16	3.04	3.06	4.09 (0.17)	3.10	**2.86 (0.07)**	5.67	5.23	6.03	4.7 (0.08)	5.76	**4.38 (0.15)**
energy	2.04	1.99	1.38	0.9 (0.04)	1.62	**0.61 (0.19)**	1.8	1.66	2.09	1.72 (0.08)	2.30	**0.39 (0.02)**
kin8nm	-0.9	-0.95	-1.2	6.89 (2.85)	-1.14	**-1.40 (0.01)**	0.1	0.1	0.09	0.26 (0)	0.09	**0.06 (0.00)**
navalpropulsion	-3.73	-3.8	-5.63	-3.03 (0.49)	-5.73	**-8.24 (0.01)**	0.01	0.01	**0**	0.01 (0)	**0.00**	**0.00 (0.00)**
powerplant	2.84	2.8	2.79	5.24 (0.72)	2.83	**2.66 (0.05)**	4.12	4.02	4.11	4.38 (0.02)	4.10	**3.43 (0.08)**
protein	2.97	2.89	2.83	21.12 (2.52)	3.12	**2.48 (0.03)**	4.73	4.36	4.71	5.09 (0.01)	4.98	**3.52 (0.02)**
wine	0.97	**0.93**	0.94	1.92 (0.92)	1.15	1.00 (0.08)	0.64	**0.62**	0.64	0.73 (0.01)	0.65	**0.62 (0.01)**
yacht	1.63	1.55	1.18	0.48 (0.19)	0.76	**-0.17 (0.11)**	1.02	1.11	1.58	1.86 (0.15)	1.00	**0.44 (0.06)**
yearprediction	3.6	3.59	3.35	7.44 (0.08)	3.58	**3.22 (0.00)**	8.88	8.85	8.89	16.84 (0.08)	9.30	**7.91 (0.04)**
Mean Rank	4.9	3.4	2.5	4.7	3.9	**1.5**	3.7	2.6	3.8	4.6	4	**1.3**

Table 6.1: Results of the regression benchmark on 10 datasets.

rior to all other methods. AutoDEUQ (AgEBO) achieves more robust estimates than those of AutoDEUQ (AgE-Mixed) and AutoDEUQ (BO-Mixed). The estimates of epistemic uncertainty for AutoDEUQ (AgEBO), AutoDEUQ (AgE-Mixed), and AutoDEUQ (BO-Mixed) show a growing trend in the interpolation region as we move away from the training region. AutoDEUQ (BO-Mixed) has larger epistemic uncertainty in the interpolation region than AutoDEUQ (AgEBO) and AutoDEUQ (AgE-Mixed) have.

The observed differences between the search methods can be attributed to the neural network diversity in the ensembles. To demonstrate this, we computed the architecture diversity for each method as follows. Each architecture was embedded as a vector of integers where each integer represents a choice for one of the decision variable of the neural architecture search space. To compute the diversity of an ensemble, we computed the

pairwise Euclidean distance between the embeddings of the architectures composing the ensemble. Then, we kept only the upper triangle of the pairwise distance matrix (because it is symmetric) and normalized it by its norm. We then computed the cumulative sum of the elements of this normalized triangular matrix, which gives us a scalar value representing diversity. AutoDEUQ (RS-Mixed) achieved the lowest diversity score (1.41), which also correlates with its poor epistemic uncertainty estimation. While AutoDEUQ (RS-Mixed) obtained diverse models for the catalog, they are not high-performing, and consequently the ensemble did not have diverse models. AutoDEUQ (AgE-Mixed) achieved a diversity score of 2.86, which resulted in a better epistemic uncertainty estimate in the interpolation region, but the estimates are poor in the extrapolation region. With a diversity score of 3.49, AutoDEUQ (BO-Mixed) obtained more diverse models, but they contributed to overly large epistemic uncertainty in the interpolation region and extrapolation regions. AutoDEUQ (AgEBO) achieved a diversity score of 3.17, which was in between that of AutoDEUQ (AgE-Mixed) and AutoDEUQ (BO-Mixed). Moreover, we found that the learning rate values obtained by AutoDEUQ (BO-Mixed) are more diverse than those obtained by AutoDEUQ (AgEBO).

6.3.3 . Quantitative assessement on regression benchmarks

Here we compare our AutoDEUQ method with probabilistic backpropagation (PBP), Monte Carlo dropout (MC-Dropout), deep ensemble (Deep Ens.), distribution-free ensembles (DF-Ens.), and hyper ensemble (Hyper Ens.) methods. While PBP is selected as a candidate for Bayesian neural network, MC-Dropout was selected for its popularity and simplicity. The Deep Ens. (with random initialization of weights, fixed architecture, and hyperparameters) will serve as a baseline method. The Hyper Ens. (ensemble with the same architecture but with different hyperparameters) is selected because it was a recently proposed high-performing ensemble method.

To assess the quality of uncertainty quantification methodologies, we used 10 regression benchmark datasets (Table 6.2) from the literature (Lakshminarayanan et al., 2017; Hernández-Lobato and Adams, 2015; Gal and Ghahramani, 2016b). We compare these methods using two metrics: (1) negative log likelihood (NLL) (*i.e.*, how likely the data is to be generated by the predicted normal distribution) and (2) root mean square error (RMSE). These two metrics were widely adopted in the literature to compare the quality of uncertainty estimation. The metric values of PBP, MC-Dropout, Deep Ens., and DF-Ens. are copied from their corresponding papers (Hernández-Lobato and Adams, 2015; Gal and Ghahramani, 2016b; Lakshminarayanan et al., 2017; Pearce et al., 2018), respectively. Nevertheless, we extended and ran the Hyper Ens. method for regression based on the information provided in (Wenzel et al., 2021).

For each dataset, we ran AgEBO to generate a catalog of 500 models and used the *greedy* selection strategy to construct ensembles of $K = 5$ members. We repeated the experiments 10 times with different random seeds for the training/validation split and computed the mean score and its standard error. An exception was the *yearprediction* dataset, which was run only 3 times because the dataset size was large.

The results are shown in Table 6.1. We observe that AutoDEUQ obtains superior per-

Dataset's Name	Number of Samples	Feature Size
boston	506	13
concrete	1030	8
energy	768	8
kin8nm	8192	8
navalpropulsion	11934	16
powerplant	9568	4
protein	45730	9
wine	1599	11
yacht	308	6
yearprediction	515345	90

Table 6.2: Description of the different datasets used in the regression benchmark.

formance compared with the other methods with respect to both NLL and RMSE. We computed the ranking of the methods for each dataset and computed the mean across the 10 datasets. This is shown in the last row of Table 6.1. AutoDEUQ with Greedy outperforms all of the other methods on 8 out of 10 datasets. On boston and wine, Hyper Ens. and MC Dropout have the lowest NLL and RMSE values. We note that, overall, the recently proposed Hyper Ens. performs worse than all the other methods. This performance can be attributed to the architecture used for regression in Hyper Ens., which is a simple multilayer perception network as described in the original paper (Wenzel et al., 2021). This further emphasizes the importance of and need for the architecture search for different datasets.

6.4 . Conclusion

We developed AutoDEUQ, an approach to automate the generation of deep ensembles for uncertainty quantification. We empirically demonstrated that epistemic uncertainty is best captured when the neural networks considered in the ensemble are diverse (in hyperparameters and architecture), yet all the neural networks perform well and similarly on the validation set. This result is achieved by a two-step process: (1) using aging evolution and Bayesian optimization to jointly explore the neural architecture and hyperparameter space and generate a diverse catalog of models and (2) using greedy selection of models optimized with the negative log likelihood, to find models that are very different but all with high (and similar) performance. We conducted an extensive regression benchmark to compare AutoDEUQ with different classes of UQ methods, with and without ensembles. Our results confirm quantitatively what was observed on the toy example. The key ingredient of our technique is the diversity and predictive strength and homogeneity of the final ensemble.

Using a toy example, we performed an ablation study to visualize the impact of different components of AutoDEUQ on uncertainty estimation. This impact appears clearly in regions depleted in the training samples. Compared with AutoDEUQ, methods optimizing

either hyperparameters independently or architecture search underestimate epistemic uncertainty. Moreover, we conducted an extensive regression benchmark study to compare AutoDEUQ against different classes of UQ methods, with and without ensembles. Our results confirm quantitatively what was observed on the toy example.

The key ingredient of our technique is the diversity and predictive strength and homogeneity of the final ensemble. AutoDEUQ is a computationally expensive method. However, the computational need can be controlled by restricting the search space and running model evaluations in parallel.

Our future work will include (1) applying AutoDEUQ on larger datasets to assess its scalability, (2) evaluating AutoDEUQ on a classification benchmark, and (3) seeking theoretical insights into the quality of epistemic uncertainty under the various data generation assumptions.

6.5 . Acknowledgement

This research was conducted in collaboration with Romit Maulik, Krishnan Raghavan, Bethany Lusch, Isabelle Guyon, and Prasanna Balaprakash. This material is based upon work supported by the U.S. Department of Energy (DOE), Office of Science, Office of Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility. This material is based upon work supported by ANR Chair of Artificial Intelligence HUMANIA ANR-19-CHIA-0022 and TAILOR EU Horizon 2020 grant 952215.

7 - Applications and influence of DeepHyper

This chapter provides an overview of the application of algorithms and techniques from this thesis, showcasing their use across different domains such as climate science, traffic forecasting, and high-performance computing (HPC). The surveyed applications are not exhaustive but highlight the breadth of fields that benefit from optimizing learning workflows through DeepHyper's capabilities in hyperparameter optimization, neural architecture search, and uncertainty quantification.

7.1 . Application to computational fluid dynamics

In this section, our focus is on a series of studies that employed DeepHyper for computational fluid dynamics. Our paper Maulik et al. (2020) addresses the development of surrogate models for geophysical forecasting, which is key for economic planning, disaster management, and climate change adaptation. Traditional methods rely on costly numerical simulations, prompting a shift towards data-driven models, specifically neural networks, for more efficient forecasting. However, creating effective neural networks for this purpose is challenging and often involves trial and error.

This research introduces a method for automatically generating proper-orthogonal-decomposition-based long short-term memory networks (POD-LSTMs) to forecast sea-surface temperature using the NOAA Optimum Interpolation Sea-Surface Temperature dataset[1]. By leveraging neural architecture search (NAS), the study aims to optimize stacked LSTM architectures without human intervention, surpassing manually designed models and baseline methods in forecasting accuracy. The approach uses the package DeepHyper (Balaprakash et al., 2018a) we developed along this thesis (Chapter 6), a scalable, open-source NAS framework, to parameterize the search space of stacked LSTM architectures as a directed acyclic graph.

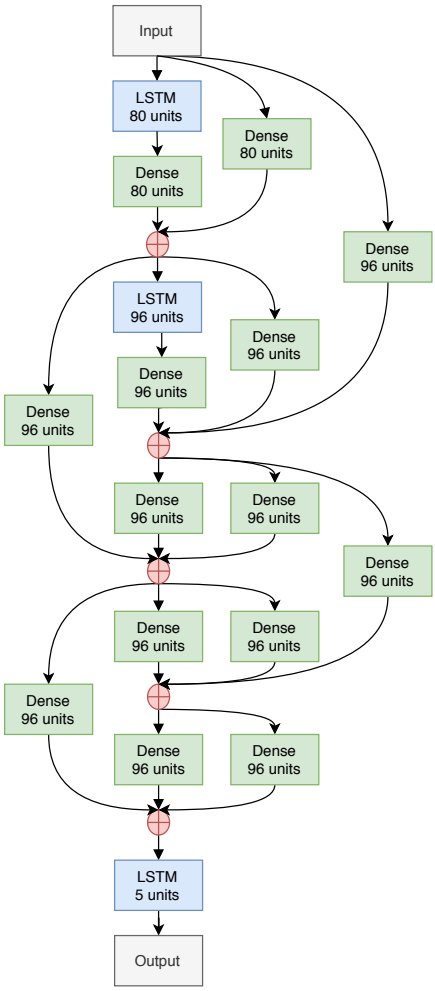
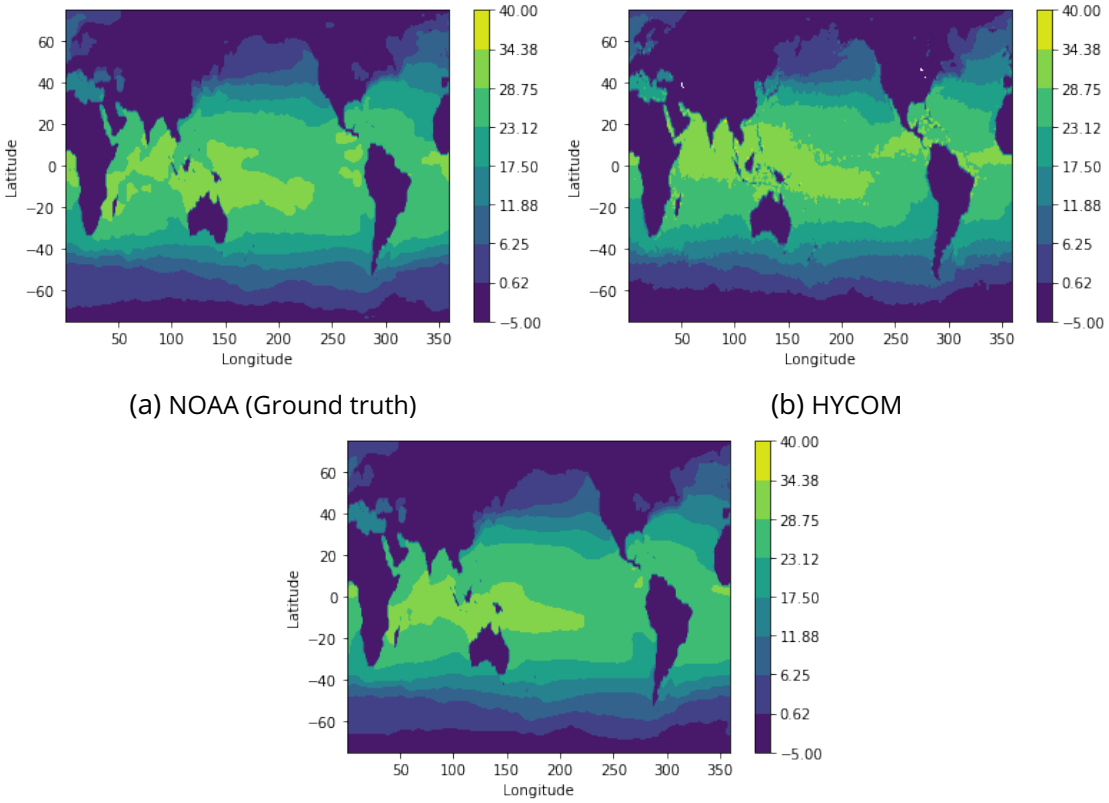


Figure 7.1: Illustration from Maulik et al. (2020) showcasing the best RNN architecture for sea-surface temperature forecasting.

The NAS-derived POD-LSTMs showed superior performance compared to manually designed models and baseline machine learning forecast tools. The method's scalability was validated on up to 512 nodes of the Theta supercomputer. An example prediction from the POD-LSTM is presented in Figure 7.2.

The machine learning problem is formulated with input as the historical sequence of sea-surface temperatures and output as the future temperature forecast. The loss function used is the mean squared error during training, with the coefficient of determination

[1]NOAA Optimum Interpolation (OI) SST V2: https://psl.noaa.gov/data/gridded/data.noaa.oisst.v2.html (last accessed March 2024)



(a) NOAA (Ground truth)

(b) HYCOM

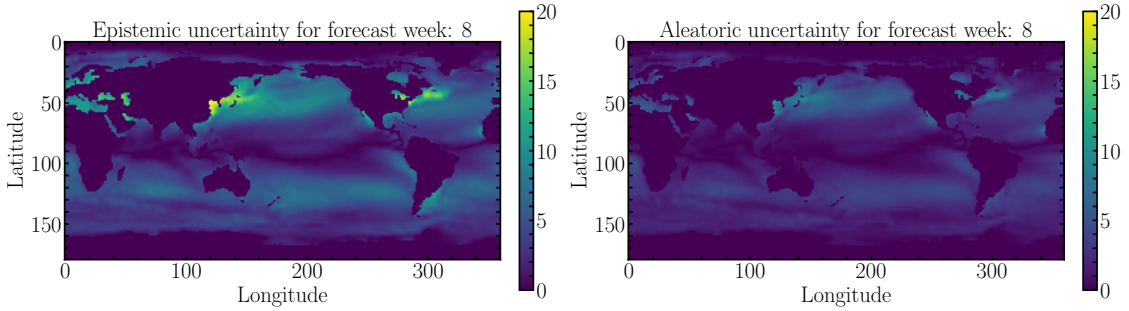
(c) POD-LSTM

Figure 7.2: Example result from Maulik et al. (2020) showcasing the ground truth data (7.2a), a state-of-the-art forecasting model (7.2b), and the prediction of the optimized POD-LSTM (7.2c) for sea-surface temperature forecasting on the week of June 14, 2015.

(R^2) as the evaluation metric on the validation dataset. The NAS effectively navigates the architecture search space, optimizing for architectures that yield high R^2 values, indicating strong predictive performance.

In a follow-up paper (Maulik et al., 2023) we extended the neural architecture search (NAS) framework for sea-surface temperature prediction via LSTM by (1) optimizing jointly training hyperparameters and (2) integrating an ensemble method to assess uncertainties. The addition of an ensemble method marks a significant advancement in the framework's capability to evaluate predictive uncertainties, addressing a crucial aspect of forecasting accuracy and reliability. Furthermore, they employed the variance-decomposition technique to discern prediction errors related to model bias or data noise. An example of the estimated epistemic and aleatoric uncertainties is shown in Figure 7.3.

The influence of the DeepHyper package has extended to a diverse array of research endeavors at Argonne National Laboratory. Specifically, (Maulik et al., 2021) leveraged the parallel Bayesian optimization (Chapter 3 in DeepHyper for the hyperparameter optimization of LSTMs and neural ordinary differential equations (NODE), addressing the temporal



(a) Epistemic Uncertainty

(b) Aleatoric Uncertainty

Figure 7.3: Example result from Maulik et al. (2023) showcasing the estimated epistemic (7.3a) and aleatoric (7.3b) uncertainty of sea-surface temperature predictions.

dynamics of the viscous Burgers equation. This optimization process not only elevated the models' predictive accuracy but also facilitated comparative assessments across different modeling approaches.

In a separate study, (Liu et al., 2022) applied DeepHyper to refine the hyperparameters of an LSTM model designed to simulate coarse mesh turbulence for transient analysis concerning thermal mixing and stratification in sodium-cooled fast reactors. The use of DeepHyper enhanced the model's predictive capabilities and contributed to a better understanding of the model's analytical performance. These instances underscore DeepHyper's broad applicability and its instrumental role in advancing computational modeling and analysis in fields beyond those directly explored by its original authors.

Overall, the combination of methods provided by DeepHyper and their application on computational fluid dynamics has significantly improved the predictive accuracy of models, facilitated their development, and enabled the quantification of predictive uncertainties. This development enhances predictive and modeling functionalities in the domain of climate science and associated disciplines.

7.2 . Application to traffic forecasting

In this section, we examine several studies utilizing DeepHyper for traffic forecasting.

Mallick et al. (2020) addresses the challenge of forecasting short-term traffic flow across large highway networks, a critical component for enhancing urban mobility and planning, notably in traffic congestion mitigation. The task presents considerable difficulty due to (1) the spatial-temporal dependencies within traffic patterns and (2) the size of the data, which traditional forecasting methodologies struggle to model. The authors use a Diffusion Convolutional Recurrent Neural Network (DCRNN) integrated with graph partitioning to enhance computational and memory efficiency for large highway networks. The example use case is on California's network with 11,160 sensor (nodes) locations. In Figure 7.4 we present an illustration from the original paper to showcase the graph-partitioning of the highway network in Los Angeles (USA).

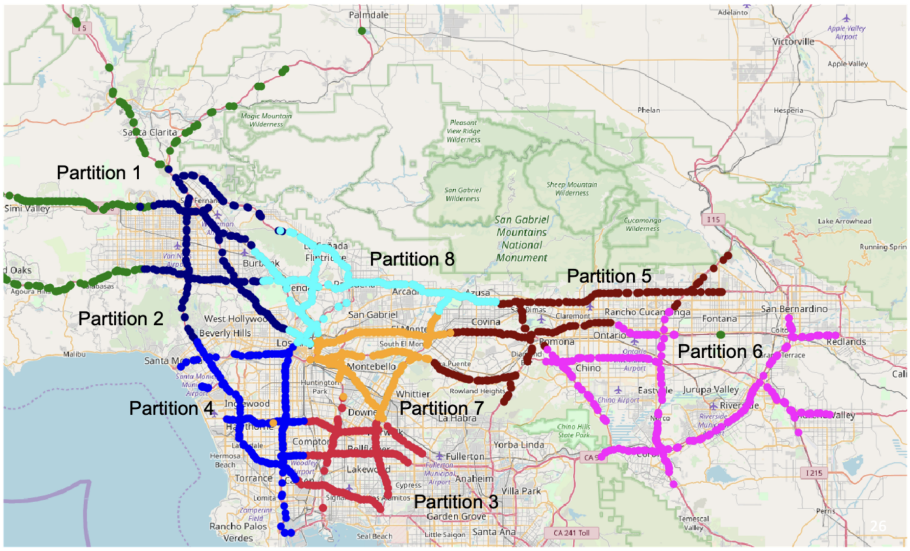


Figure 7.4: Illustration of graph-partitioning of the high-way network in Los Angeles (USA) from Mallick et al. (2020).

The learning is based on historical data from distributed sensors throughout the high-way network to forecast future traffic conditions (*e.g.*, vehicle speed and flow). Input to the DCRNN consists of time-series graph representations of the highway network, where nodes correspond to sensors and edges to the road segments connecting these sensors. The DCRNN output is a predictive time series of short-term traffic conditions for each network sensor. Predictive accuracy is quantified using the mean absolute error (MAE).

To enhance the model's accuracy the study employs DeepHyper for parallel hyper-parameter optimization (Chapter 3), concentrating on hyperparameters such as learning rate, number of layers, units per layer, and graph filter type. Empirical outcomes indicate superior prediction accuracy through the application of our hyperparameter optimization technique over manual tuning.

A further investigation by Mallick et al. (2022) applies our parallel hyperparameter optimization framework to build an ensemble of deep neural networks aimed for predictive uncertainty, aligning closely with our methodologies from Chapter 6. To avoid repeated hyperparameter optimization, a generative model is derived from the set of best-observed candidates after hyperparameter optimization, facilitating the on-demand generation of diverse and accurate candidates to build new ensembles.

Another work Sun et al. (2022), including Mallick T. and Balaprakash P. as common co-authors, focuses on the identification of traffic incidents. Instead of developing a new model, this work focuses on improving the quality of training data. It was noticed by the authors that previously trained models through supervised learning had a high false alarm rate. After this observation, it was identified that training data (similar to previous works on DCRNN) contained mislabeled incidents that explained the difficult learning. Therefore the authors proposed a data-centric framework based on weak supervised

learning to improve label quality. Weak supervision corresponds to using a combination of heuristic rules to attribute probabilistic labels to input data. The best model obtained from this weak supervision is an RNN for which the hyperparameters were optimized using DeepHyper's parallel Bayesian optimization.

Overall, the application of DeepHyper in traffic forecasting has helped in the development of more accurate predictive models. However, as demonstrated by the last work, improving the quality of training data is also crucial for the success of machine learning models.

7.3 . Application to the optimization of HPC software

In this final application section, we examine works utilizing DeepHyper methods to optimize HPC software performance. Although the thesis focus excludes high-performance computing (HPC) software optimization, algorithms developed for hyperparameter optimization in learning workflows can be used in this domain.

Our study with Dorier et al. (2022) tackles the task of automatically optimizing parameters for an HPC storage service to enhance I/O and storage performance, a key challenge within the HPC community. Due to similar but still different deployment settings of the optimized software, we propose a transfer-learning approach for hyperparameter optimization. This method extends our parallel Bayesian optimization (Chapter 3) technique by leveraging prior optimization outcomes to better condition the initialization of new optimization processes. Specifically, we use a generative model from optimal parameters of past runs to suggest initial samples for a new optimization. An example result of this transfer-learning strategy is presented in Figure 7.5. Integrated into the DeepHyper framework, our approach undergoes testing against the optimization of a specialized I/O service for high-energy physics workflows on Argonne's Theta supercomputer, showcasing a substantial speed enhancement facilitated by transfer learning.

Another of our papers by Dash et al. (2023) delves into the challenge of training large language models (LLMs) on high-performance computing (HPC) systems. The challenge of this work is to efficiently scale up model training from billions to trillions of parameters, necessitating solutions to address memory constraints and communication latency for effective distributed training.

To tackle memory constraints, the paper explores techniques for fitting large models into GPU memory by distributing them across multiple GPUs. Strategies such as pipelining, tensor parallelism (also known as model parallelism), and data parallelism are discussed to effectively manage the model's memory footprint.

Regarding communication latency, the paper investigates methods enabling overlap between computation and communication, thus minimizing idle times and enhancing overall training efficiency.

The integration of these strategies provides a range of parameters for overall training time. For instance, tensor parallelism determines how model layers are divided across GPUs, crucial for balancing computational load and reducing communication overhead.

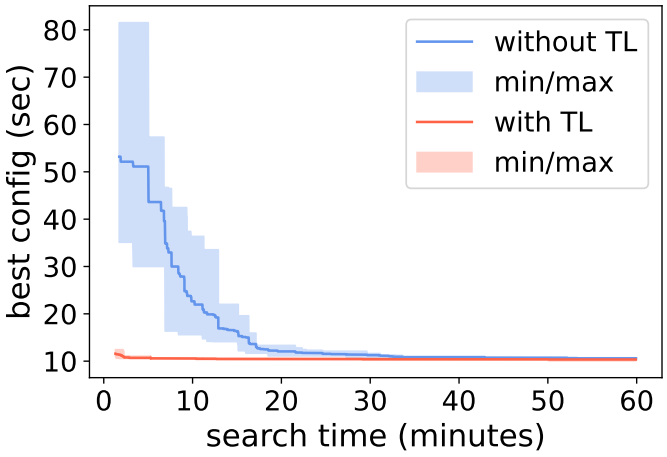


Figure 7.5: Performance profile from Dorier et al. (2022) where the goal is to minimize the run-time in seconds of the best parameter configuration (y-axis). The blue curve represents a standard optimization without transfer-learning while the red curve represents the optimization with transfer-learning. The x-axis represents the execution time of the optimization process. This result illustrates the significant speed-up enabled by transfer learning.

Pipelining involves segmenting the model into stages for sequential processing, impacting throughput and latency. Data parallelism, on the other hand, is influenced by the number of microbatches, affecting the granularity of training steps. Thus, directly impacting the efficiency of overlapping computation with communication. Additionally, the global batch size affects training speed and convergence, requiring meticulous tuning to preserve model accuracy while leveraging parallelism.

Furthermore, employing mixed precision aids in reducing memory requirements and increasing computational throughput.

The study underscores the efficacy of parallel and asynchronous hyperparameter optimization with DeepHyper in identifying efficient strategies for training LLMs of various sizes on exascale HPC systems. Moreover, it demonstrates the potential to achieve significant enhancements in computational performance and scaling efficiency.

Overall the application of DeepHyper in HPC software optimization has helped replace the tedious manual task of minimizing software run-time. The parallel asynchronous optimization capabilities of DeepHyper were key enabling factors to the success of these projects. In particular, the adaptability of DeepHyper's software architecture, which facilitates seamless parallelization with various backends such as thread pools, process pools, or MPI, explains its adoption and effectiveness.

7.4 . Conclusion

This chapter has provided an overview of the application of algorithms and techniques from this thesis, showcasing their use across different domains such as climate science, traffic forecasting, and high-performance computing (HPC). The surveyed applications are not exhaustive but highlight the breadth of fields that benefit from optimizing learning workflows through DeepHyper's capabilities in hyperparameter optimization, neural architecture search, and uncertainty quantification. Other applications of DeepHyper that were not detailed in this Chapter include optical character recognition (Belay et al., 2023), molecular and crystal properties (Park et al., 2023), and continual learning (Madireddy et al., 2023).

The scalability of the developed algorithms was an essential factor for these different projects to obtain desired outcomes in a reasonable time. Without the parallel and asynchronous capabilities of DeepHyper, the optimization of hyperparameters and neural architectures would have been practically infeasible for these applications. The use of hyperparameter optimization provided improved predictive performance in most applications. However, it could not solve issues related to the quality of training data, as demonstrated in the traffic incident detection application. This suggests that the quality of the data is as important as the quality of the model for the success of machine learning applications. Therefore, automated data-quality checks should be combined with automated model optimization to ensure the success of machine learning applications.

8 - General conclusion and future perspectives

8.1 . Conclusion on the optimization of learning workflows at large scale

The fundamental research objective of this thesis aimed to devise scalable methodologies for efficiently and effectively optimizing learning workflows on parallel high-performance computing systems, also known as supercomputers. Specifically, we focused on investigating deep neural network learning workflows. This section provides a concise overview of the findings derived from this research endeavor.

In **Chapter 3**, the primary research objective was to develop an efficient and highly parallelizable optimization algorithm for the optimization of learning workflows. Our focus centered on Bayesian optimization due to its recognized efficacy in optimizing resource-intensive functions. We expanded upon the conventional sequential Bayesian optimization approach to render it parallel, asynchronous, and decentralized, thereby addressing a fundamental bottleneck of previously proposed centralized Bayesian optimization methods.

Subsequently, we empirically validated the scalability of our algorithm by deploying it across 1,920 parallel processes, each utilizing 1 GPU, and in a preceding study involving 4,096 processes, each utilizing 16 CPU cores. Our findings underscored the sustained efficiency of our algorithm across varying levels of parallelization. This translates practically into two key benefits: firstly, alleviating user concerns regarding the optimal parallel scale for hyperparameter optimization; and secondly, ensuring that the outcomes of hyperparameter optimization are improved by scaling, a non-trivial accomplishment as many optimization algorithms experience degradation under increased scaling due to bottlenecks (*e.g.*, as seen in the case of Bayesian optimization with Gaussian processes).

Before our work, no comprehensive investigation into large-scale parallel Bayesian optimization had been conducted in the literature. Our research serves to inform practitioners about the advantages (*e.g.*, speed-up) and constraints (*e.g.*, potential stagnation) associated with large-scale parallelism in hyperparameter optimization. Subsequent chapters of this thesis build upon the foundation laid by this decentralized Algorithm 2.

In **Chapter 4**, we addressed the single-objective limitation of our decentralized Algorithm 2 to make it compatible with multiple objectives. This adaptation became necessary due to the diverse requirements that modern learning workflows must fulfill, such as predictive performance, social fairness, latency, and memory consumption.

To achieve this, in Algorithm 3, we introduced a randomized scalarization method coupled with (1) a quantile-based normalization of objectives and (2) a soft penalty mechanism to discourage exploration of uninteresting objective trade-offs. The proposed normalization is robust to outliers while conserving the optimal solution set, and helps different objectives to be on the same scale, thereby facilitating the scalarization process. The proposed normalization method is robust to outliers while preserving the optimal solution set. Additionally, it facilitates the scalarization process by ensuring that different

objectives are mapped to comparable scales. Prior works on hyperparameter optimization used manually defined objective normalization and outlier filtering processes which requires more domain knowledge about optimized objectives. The soft penalty mechanism, on the other hand, discourages the exploration of uninteresting objective trade-offs, thereby making the optimization process more effective in practice. We are the first to explore the use of soft penalties with a Random-Forest-based Bayesian optimization algorithm.

Empirical evaluations demonstrate that this approach outperforms other Bayesian optimization techniques as well as a state-of-the-art genetic algorithm (NSGAII). Consistent with the findings in Chapter 3, our multi-objective algorithm exhibits superior efficiency across all tested scales, ranging from 40 to 640 parallel processes.

In **Chapter 5**, the aim was to speed up the optimization process by mitigating the repetitive costs associated with lengthy deep neural network training. Our approach involved investigating methods that could seamlessly extend Algorithm 2. Consequently, we benchmarked early discarding techniques' behavior when observing a stream of "learning curves" from various hyperparameter configurations. We conducted benchmarks on prominent early discarding methods, namely Successive Halving and Learning Curve Extrapolation, along with a straightforward baseline (i-Epoch) that halts training after i epochs. Our benchmarking framework adopted a multi-objective perspective, aiming to (1) minimize the final prediction error and (2) minimize the overall number of training epochs utilized. Our experiments revealed that selecting based on a single epoch training often achieves performance close to optimality while maintaining simplicity. This implies the existence of hyperparameter configurations that outperform others regardless of the training epoch. Furthermore, we observed that employing a fixed number of epochs yields comparable or superior results to utilizing more complex (*i.e.*, dynamic) early discarding methods. We confirmed these findings by assessing the empirical Pareto fronts of each early discarding method being investigated.

In **Chapter 6**, we leveraged our algorithms for optimizing learning workflows as a foundational capability to enhance the quantification of uncertainty in deep neural networks. Deep ensembles are recognized for their competitiveness in uncertainty quantification research and their role as variance-reduction methods for machine learning, rendering them more resilient to the inherent randomness stemming from the dataset or the learning process. In our study, we explored the use of evaluated learning workflows from hyperparameter optimization as the basis for constructing an ensemble of deep neural networks. This approach demonstrated an enhanced estimation of epistemic uncertainty (arising from the model), attributed to the increased diversity among explored models. Moreover, it streamlined the manual tuning of hyperparameters to achieve accurate estimations of both point-wise predictions (*i.e.*, mean) and aleatoric uncertainty (arising from the data-generating process).

In summary, our series of contributions empower machine learning practitioners to (1) efficiently automate the optimization of learning workflows, spanning from small-scale laptops to the most advanced supercomputers developed to date (Dash et al., 2023), and (2) serves as a novel capability for fostering further advancements in machine learning

algorithm design, as illustrated in Chapter 6. Our work draws significant inspiration from the renowned SMAC algorithm (Hutter et al., 2011; Lindauer et al., 2022). All the contributions presented are accessible through the open-source DeepHyper Python Package (Balaprakash et al., 2018a).

8.2 . Future perspectives on the optimization of learning workflows

Numerous research opportunities exist to enhance the optimization of learning workflows, spanning across both small and large scales. Here, we outline several avenues for pursuing these improvements.

Transfer and Meta-Learning: From a machine learning perspective, a key limitation of our research is the absence of iterative learning across optimization tasks, commonly referred to as meta-learning (from many tasks to many tasks) (Ruhkopf et al., 2022) or transfer-learning (from one task to another). Incorporating such approaches could enhance the optimization process during initialization and prevent often redundant initial exploration phases across optimization processes. Although we did not specifically investigate this aspect within the context of learning workflows, we proposed a straightforward transfer-learning mechanism that demonstrated effectiveness in optimizing the runtime of an HPC database service (Dorier et al., 2022).

Interpretability and Explainability: The interpretability and explainability of machine learning models are critical for ensuring the trustworthiness and accountability of automated decision-making systems. In this work, we focused on optimizing learning workflows, with a particular emphasis on predictive performance. Future research could explore the use of quantitative metrics for interpretability and explainability to optimize them jointly with predictive performance. Therefore enabling the identification of hyperparameter configurations that yield not only high predictive performance but also models that are interpretable and explainable. Another aspect would be to develop methods that explain the hyperparameter optimization process itself, providing insights into why certain hyperparameter configurations were selected over others. This is particularly challenging due to the dependence between iterations of the process. During our research, we resorted to using the SandDance tool[1] (Drucker and Fernandez, 2015) to visualize the hyperparameter optimization process, but more formal methods could be developed.

Overfitting with small datasets: Throughout this research, we employed the widely utilized three-way split approach, encompassing training, validation, and test sets, to facilitate model selection (Section 2.2.2). This method is customary in hyperparameter optimization for deep neural networks. Its viability hinges on the availability of datasets with a sufficient volume of data, allowing the estimation of performance on the validation set to serve as a reliable approximation of performance on unseen test data. In instances where datasets are limited in size, implementing more robust model selection strategies becomes imperative. For example, employing K-fold cross-validation can offer a solution. However, this alternative approach can be computationally expensive as it necessitates

[1]SandDance: https://microsoft.github.io/SandDance/ (accessed March 2024)

repeated training of models. Despite this drawback, it provides a more thorough assessment of model performance across different subsets of the data, mitigating the risk of overfitting and yielding more reliable results.

Bayesian optimization: Benchmarking the components of Bayesian optimization algorithms poses several challenges due to the intricate nature of each element and their interdependencies. Evaluating the surrogate model's performance, which approximates the objective function, requires careful consideration, especially as the performance observations collected are not independent and identically distributed. Additionally, assessing the surrogate model's implementation introduces complexities, as variations in algorithms and hyperparameters significantly influence performance (Section 3.2.1). Moreover, the acquisition function, crucial for guiding the optimization process, poses another layer of difficulty, as different formulations and strategies for utilizing uncertainty must be evaluated. Furthermore, selecting an appropriate optimizer for minimizing the acquisition function presents challenges, with the choice depending on factors such as the surrogate model and the acquisition functions themselves. Benchmarking these components under various conditions is essential for understanding their effectiveness and robustness.

HPC system performance issues: From an HPC standpoint, a significant limitation is the lack of detailed insight into the resource utilization of learning workflows. For instance, precise metrics concerning memory and compute usage (*e.g.*, GPU utilization) were not provided. We anticipate that such metrics would reveal (1) redundant memory consumption (*e.g.*, multiple duplicates of the same data) and (2) suboptimal utilization of computational resources. Additionally, we foresee potential I/O latencies arising from the simultaneous loading of large datasets by multiple processes, leading to delays when accessing the filesystem.

Ethical and Fair AI: The ethical implications of machine learning algorithms are a growing concern, for example with the potential for social biases to be inadvertently encoded into models. Incorporating fairness metrics into the multi-objective optimization process could help mitigate these biases. Also identifying the hyperparameters that influence this fairness could be a research direction. For example, the class weights could be optimized to explore the trade-off between predictive performance and fairness.

AI and Sustainability: The environmental impact of machine learning workflows presents a critical concern, especially given the field's continuous expansion and escalating resource consumption. One promising avenue of research involves optimizing hyperparameters to minimize resource utilization of learning workflows. Nevertheless, the consumption of High-Performance Computing (HPC) systems cannot be neglected. This study introduced the feasibility of reducing the number of training epochs (see Chapter 5) during the optimization process; thereby substantially decreasing the total training epoch count by a factor of 40. Thus reducing overall resource utilization. Further investigations along this trajectory could explore additional strategies such as reducing the volume of training samples. Also reducing overall resource utilization. Additionally, employing meta-learning techniques can alleviate the initialization overheads of this process by providing better early-stage suggestions; thus mitigating inefficient and resource-intensive explo-

ration.

A - Formal Notations

In this appendix, we provide a summary of the formal notations we used.

- $\mathbb{R}$ set of real numbers.
- $\mathbb{Z}$ set of discrete numbers.
- $\mathbb{N}$ set of natural numbers.
- $\mathbb{K}$ a set of categorical values.
- $:=$ definition.
- $=$ equality.
- $\approx$ approximation.
- $[|a, b|]$ a discrete range.
- $[a, b]$ a continuous range.

A.1 . Probability and Statistics

- *i.i.d.* : independantly and identically distributed.
- upper case X are random variables.
- lower case x are regular variables.
- $\mathrm{P}(X)$ probability distribution/law (the input is a random variable).
- $\mathrm{p}(x)$ probability density (*p.d.f.*) or probability mass (*p.m.f.*) functions (the input is a classic variable).
- $\mathrm{F}(x)$ cumulative distribution function (*c.d.f.*) (the input is a classic variable).
- $\varphi(.)$ distribution function of the standard normal law.
- $\Phi(.)$ cumulative distribution function of the standard normal law.
- $\mathbb{E}[.]$ expectation operator.
- $\mathbb{V}[.]$ variance operator.
- $\mathbb{I}(X; Y)$ mutual-information operator.
- R^2 coefficient of determination.

A.2 . Learning Theory

- $D = \{(X_1, Y_1), \ldots, (X_n, Y_n)\}$ is a dataset of n samples.
- $(x, y) \in (\mathcal{X}, \mathcal{Y})$ are input and target data.
- Θ is hyperparameter search space (possibly inlcuding preprocessing, training and neural architecture hyperparameters).
- $\theta \in \Theta$ is a hyperparameter vector.
- Θ is a hyperparameter random vector.
- $\mathcal{A}_\theta$ is a set of predictor configured with hyperparmeters θ.
- $\alpha_\theta \in \mathcal{A}_\theta$ is a predictor configured with hyperparameters θ.
- A_θ is a predictor random variable configured with hyperparameters θ.
- $\mathcal{B}_\theta$ is a set of learner algorithms configured with hyperparameters θ.
- $\beta_\theta \in \mathcal{B}_\theta$ is a learner configured with hyperparameters θ.
- B_θ is a learner random variable configured with hyperparameters θ.
- $L(\ldots) \in \mathbb{R}$ is a loss function.
- $R(.)$ is the risk.
- $R_{\text{emp}}(.)$ is the empirical risk.

A.3 . Optimization

- $\mathcal{C}$ is a cost or objective space.
- $c \in \mathcal{C}$ is a cost vector (*e.g.*, loss, time, FLOPS, etc.).
- $f \in \mathcal{F}$ is the cost function to optimize.
- $g \in \mathcal{G}$ is the constraint function.
- x^* is an optimal solution.

Bibliography

S. Adriaensen, H. Rakotoarison, S. Müller, and F. Hutter. Efficient bayesian learning curve extrapolation using prior-data fitted networks. *arXiv preprint arXiv:2310.20447*, 2023.

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*, pages 2623–2631, 2019a.

T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A next-generation hyperparameter optimization framework, 2019b. URL https://arxiv.org/abs/1907.10902.

A. Alvi, B. Ru, J.-P. Calliess, S. Roberts, and M. A. Osborne. Asynchronous batch Bayesian optimisation with improved local penalisation. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 253–262. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/alvi19a.html.

D. Amaratunga and J. Cabrera. Analysis of data from viral dna microchips. *Journal of the American Statistical Association*, 96(456):1161–1170, 2001.

A. Ashukha, A. Lyzhov, D. Molchanov, and D. Vetrov. Pitfalls of in-domain uncertainty estimation and ensembling in deep learning. *arXiv preprint arXiv:2002.06470*, 2020.

S. Astete-Morales, M.-L. Cauwet, J. Liu, and O. Teytaud. Simple and cumulative regret for continuous noisy optimization. *Theoretical Computer Science*, 617:12–27, 2016.

C. Audet and J. E. Dennis. A progressive barrier for derivative-free nonlinear programming. *SIAM Journal on Optimization*, 20(1):445–472, 2009. doi: 10.1137/070692662.

C. Audet, J. Bigeon, D. Cartier, S. Le Digabel, and L. Salomon. Performance indicators in multiobjective optimization. *European journal of operational research*, 292(2):397–422, 2021.

N. Awad, N. Mallik, and F. Hutter. DEHB: Evolutionary hyberband for scalable, robust and efficient hyperparameter optimization. In Z. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 2147–2153. ijcai.org, 2021.

T. Bäck and H.-P. Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary computation*, 1(1):1–23, 1993.

M. Balandat, B. Karrer, D. Jiang, S. Daulton, B. Letham, A. G. Wilson, and E. Bakshy. Botorch: A framework for efficient monte-carlo bayesian optimization. *Advances in neural information processing systems*, 33:21524–21538, 2020.

P. Balaprakash, R. Egele, M. Salim, R. Maulik, V. Vishwanath, S. Wild, et al. "deephyper: A python package for scalable neural architecture and hyperparameter search", 2018a. URL https://github.com/deephyper/deephyper.

P. Balaprakash, M. Salim, T. D. Uram, V. Vishwanath, and S. M. Wild. Deephyper: Asynchronous hyperparameter search for deep neural networks. In *IEEE 25th international conference on high performance computing (HiPC)*, pages 42–51. IEEE, 2018b.

P. Balaprakash, R. Égelé, M. Salim, S. Wild, V. Vishwanath, F. Xia, T. Brettin, and R. Stevens. Scalable reinforcement-learning-based neural architecture search for cancer deep learning research. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450362290. doi: 10.1145/3295500.3356202. URL https://doi.org/10.1145/3295500.3356202.

A. Bansal, D. Stoll, M. Janowski, A. Zela, and F. Hutter. JAHS-Bench-201: A foundation for research on joint architecture and hyperparameter search. In *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2022.

A. R. Barron. Approximation and estimation bounds for artificial neural networks. *Machine learning*, 14:115–133, 1994.

T. Bartz-Beielstein. A survey of model-based methods for global optimization. *Bioinspired Optimization Methods and Their Applications*, pages 1–18, 2016.

E. Begoli, T. Bhattacharya, and D. Kusnezov. The need for uncertainty quantification in machine-assisted medical decision making. *Nature Machine Intelligence*, 1(1):20–23, 2019.

B. H. Belay, I. Guyon, T. Mengiste, B. Tilahun, M. Liwicki, T. Tegegne, and R. Egele. Hhd-ethiopic a historical handwritten dataset for ethiopic ocr with baseline models and human-level performance. 2023.

J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. 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, 2011. Curran Associates Inc. ISBN 9781618395993.

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*, pages 115–123. PMLR, 2013.

B. Bischl, J. Richter, J. Bossek, D. Horn, J. Thomas, and M. Lang. mlrmbo: A modular framework for model-based optimization of expensive black-box functions. *arXiv preprint arXiv:1703.03373*, 2017.

C. M. Bishop. Mixture density networks. 1994.

J. Blank and K. Deb. pymoo: Multi-objective optimization in python. *IEEE Access*, 8:89497–89509, 2020. doi: 10.1109/ACCESS.2020.2990567.

A. Blum and M. Hardt. The ladder: A reliable leaderboard for machine learning competitions. In *International Conference on Machine Learning*, pages 1006–1014. PMLR, 2015.

C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural network. In *International Conference on Machine Learning*, pages 1613–1622. PMLR, 2015.

O. Bohdal, L. Balles, M. Wistuba, B. Ermis, C. Archambeau, and G. Zappella. Pasha: Efficient hpo and nas with progressive resource allocation. In *ICLR*, 2023.

B. M. Bolstad, R. A. Irizarry, M. Åstrand, and T. P. Speed. A comparison of normalization methods for high density oligonucleotide array data based on variance and bias. *Bioinformatics*, 19(2):185–193, 2003.

L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer, 2010.

L. Breiman. Bagging predictors. *Machine learning*, 24:123–140, 1996.

L. Breiman. Random forests. *Machine learning*, 45:5–32, 2001.

N. Burkart and M. F. Huber. A survey on the explainability of supervised machine learning. *Journal of Artificial Intelligence Research*, 70:245–317, 2021.

R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Twenty-first international conference on Machine learning - ICML '04*, page 18. ACM Press, 2004. doi: 10.1145/1015330.1015432. URL http://portal.acm.org/citation.cfm?doid=1015330.1015432.

R. Caruana, A. Munson, and A. Niculescu-Mizil. Getting the most out of ensemble selection. In *Sixth International Conference on Data Mining (ICDM'06)*, pages 828–833. IEEE, 2006. doi: 10.1109/ICDM.2006.76. URL http://ieeexplore.ieee.org/document/4053111/. ISSN: 1550-4786.

T. H. Chang and S. M. Wild. Designing a framework for solving multiobjective simulation optimization problems. *arXiv preprint arXiv:2304.06881*, 2023a.

T. H. Chang and S. M. Wild. ParMOO: A python library for parallel multiobjective simulation optimization. *Journal of Open Source Software*, 8(82):4468, 2023b.

H. Cho, Y. Kim, E. Lee, D. Choi, Y. Lee, and W. Rhee. Basic enhancement strategies when using bayesian optimization for hyperparameter tuning of deep neural networks. *IEEE access*, 8:52588–52608, 2020.

X. Chu, X. Wang, B. Zhang, S. Lu, X. Wei, and J. Yan. {DARTS}-: Robustly stepping out of performance collapse without indicators. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=KLH36ELmwIB.

T. Chugh. Scalarizing functions in bayesian multiobjective optimization. In *2020 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2020.

G. Cocchi and M. Lapucci. An augmented lagrangian algorithm for multi-objective optimization. *Computational Optimization and Applications*, 77(1):29–56, 2020.

D. Cox and S. John. A statistical method for global optimization. In *[Proceedings] 1992 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1241–1246 vol.2, Oct 1992. doi: 10.1109/ICSMC.1992.271617.

C. Cristescu and J. Knowles. Surrogate-based multiobjective optimization: Parego update and test. In *Workshop on Computational Intelligence (UKCI)*, volume 770, page 46, 2015.

I. Das and J. E. Dennis. Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems. *SIAM Journal on Optimization*, 8(3):631–657, 1998.

S. Dash, I. Lyngaas, J. Yin, X. Wang, R. Egele, G. Cong, F. Wang, and P. Balaprakash. Optimizing distributed training on frontier for large language models. *arXiv preprint arXiv:2312.12705*, 2023.

S. Daulton, M. Balandat, and E. Bakshy. Differentiable expected hypervolume improvement for parallel multi-objective bayesian optimization. *Advances in Neural Information Processing Systems*, 33:9851–9864, 2020.

P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

N. De Freitas, A. J. Smola, and M. Zoghi. Exponential regret bounds for gaussian process bandits with deterministic observations. In *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ICML'12, page 955–962, Madison, WI, USA, 2012. Omnipress. ISBN 9781450312851.

K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE transactions on evolutionary computation*, 18(4):577–601, 2013.

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable test problems for evolutionary multiobjective optimization. In *Evolutionary Multiobjective Optimization, Theoretical Advances and Applications*, chapter 6. Springer, London, UK, 2005.

S. Deshpande, L. T. Watson, and R. A. Canfield. Multiobjective optimization using an adaptive weighting scheme. *Optimization Methods and Software*, 31(1):110–133, 2016.

E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91:201–213, 2002.

T. Domhan, J. T. Springenberg, and F. Hutter. Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves. In *Twenty-fourth international joint conference on artificial intelligence*, 2015.

P. Domingos. A unified bias-variance decomposition. In *Proceedings of 17th international conference on machine learning*, pages 231–238. Morgan Kaufmann Stanford, 2000.

M. Dorier, R. Égelé, P. Balaprakash, J. Koo, S. Madireddy, S. Ramesh, A. D. Malony, and R. Ross. Hpc storage service autotuning using variational- autoencoder -guided asynchronous bayesian optimization. In *2022 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 381–393, Los Alamitos, CA, USA, sep 2022. IEEE Computer Society. doi: 10.1109/CLUSTER51413.2022.00049. URL https://doi.ieeecomputersociety.org/10.1109/CLUSTER51413.2022.00049.

S. Drucker and R. Fernandez. A unifying framework for animated and interactive unit visualizations. *Microsoft Research*, 1, 2015.

C. Dwork. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*, pages 1–19. Springer, 2008.

R. Égelé, P. Balaprakash, I. Guyon, V. Vishwanath, F. Xia, R. Stevens, and Z. Liu. Agebo-tabular: Joint neural architecture and hyperparameter search with autotuned data-parallel training for tabular data. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384421. doi: 10.1145/3458817.3476203. URL https://doi.org/10.1145/3458817.3476203.

R. Egelé, I. Guyon, V. Vishwanath, and P. Balaprakash. Asynchronous decentralized bayesian optimization for large scale hyperparameter optimization. In *2023 IEEE 19th International Conference on e-Science (e-Science)*, pages 1–10. IEEE, 2023.

K. Eggensperger, M. Feurer, F. Hutter, J. Bergstra, J. Snoek, H. Hoos, K. Leyton-Brown, et al. Towards an empirical foundation for assessing bayesian optimization of hyperparameters. In *NIPS workshop on Bayesian Optimization in Theory and Practice*, volume 10, 2013.

K. Eggensperger, P. Müller, N. Mallik, M. Feurer, R. Sass, A. Klein, N. Awad, M. Lindauer, and F. Hutter. HPOBench: A collection of reproducible multi-fidelity benchmark problems for HPO. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021a. URL https://openreview.net/forum?id=1k4rJYEwda-.

K. Eggensperger, P. Müller, N. Mallik, M. Feurer, R. Sass, A. Klein, N. Awad, M. Lindauer, and F. Hutter. Hpobench: A collection of reproducible multi-fidelity benchmark problems for hpo. *arXiv preprint arXiv:2109.06716*, 2021b.

M. Ehrgott. *Multicriteria optimization*, volume 491. Springer Science & Business Media, 2005.

R. El-Yaniv, Y. Geifman, and Y. Wiener. The prediction advantage: A universally meaningful performance measure for classification and regression. *arXiv preprint arXiv:1705.08499*, 2017.

T. Elsken, J. H. Metzen, and F. Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

S. Falkner, A. Klein, and F. Hutter. Bohb: Robust and efficient hyperparameter optimization at scale. In *International conference on machine learning*, pages 1437–1446. PMLR, 2018.

P. I. Frazier. A Tutorial on Bayesian Optimization. *arXiv:1807.02811 [cs, math, stat]*, July 2018. URL http://arxiv.org/abs/1807.02811. arXiv: 1807.02811.

Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059. PMLR, 2016a.

Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. 2016b. URL http://arxiv.org/abs/1506.02142.

J. Garcia-Barcos and R. Martinez-Cantin. Fully distributed bayesian optimization with stochastic policies. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 2357–2363. International Joint Conferences on Artificial Intelligence Organization, 7 2019. doi: 10.24963/ijcai.2019/327. URL https://doi.org/10.24963/ijcai.2019/327.

J. Gawlikowski, C. R. N. Tassi, M. Ali, J. Lee, M. Humt, J. Feng, A. Kruspe, R. Triebel, P. Jung, R. Roscher, et al. A survey of uncertainty in deep neural networks. *Artificial Intelligence Review*, pages 1–77, 2023.

P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63: 3–42, 2006.

D. Ginsbourger, R. Le Riche, and L. Carraro. Kriging is well-suited to parallelize optimization. In Y. Tenne and C.-K. Goh, editors, *Computational Intelligence in Expensive Optimization Problems*, volume 2, pages 131–162. Springer Berlin Heidelberg, 2010a. ISBN 978-3-642-10700-9 978-3-642-10701-6. doi: 10.1007/978-3-642-10701-6_6. URL http://link.springer.com/10.1007/978-3-642-10701-6_6. Series Title: Adaptation Learning and Optimization.

D. Ginsbourger, R. L. Riche, and L. Carraro. Kriging is well-suited to parallelize optimization. In *Computational intelligence in expensive optimization problems*, pages 131–162. Springer, 2010b.

J. González, Z. Dai, P. Hennig, and N. Lawrence. Batch bayesian optimization via local penalization. In *Artificial intelligence and statistics*, pages 648–657. PMLR, 2016.

Z. Guo, X. Zhang, H. Mu, W. Heng, Z. Liu, Y. Wei, and J. Sun. Single path one-shot neural architecture search with uniform sampling. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XVI 16*, pages 544–560. Springer, 2020.

I. Guyon, A. Saffari, G. Dror, and G. Cawley. Model selection: beyond the bayesian/frequentist divide. *Journal of Machine Learning Research*, 11(1), 2010.

L. Hansen and P. Salamon. Neural network ensembles. 12(10):993–1001, 1990. ISSN 01628828. doi: $10.1109/34.58871$. URL http://ieeexplore.ieee.org/document/58871/.

M. Hauschild and M. Pelikan. An introduction and survey of estimation of distribution algorithms. *Swarm and evolutionary computation*, 1(3):111–128, 2011.

S. Heldens, P. Hijma, B. V. Werkhoven, J. Maassen, A. S. Z. Belloum, and R. V. Van Nieuwpoort. The landscape of exascale research: A data-driven literature analysis. *ACM Comput. Surv.*, 53(2), mar 2020. ISSN 0360-0300. doi: $10.1145/3372390$. URL https://doi.org/10.1145/3372390.

P. Hennig and C. J. Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13(6), 2012.

J. M. Hernández-Lobato, M. W. Hoffman, and Z. Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. *Advances in neural information processing systems*, 27, 2014.

J. M. Hernández-Lobato, J. Requeima, E. O. Pyzer-Knapp, and A. Aspuru-Guzik. Parallel and distributed thompson sampling for large-scale accelerated exploration of chemical space. In *International conference on machine learning*, pages 1470–1479. PMLR, 2017.

J. M. Hernández-Lobato and R. P. Adams. Probabilistic backpropagation for scalable learning of Bayesian neural networks. 2015. URL http://arxiv.org/abs/1502.05336.

M. D. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference. *Journal of Machine Learning Research*, 14(5), 2013.

K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

E. Hüllermeier and W. Waegeman. Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods. *Machine Learning*, 110(3):457–506, 2021.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Learning and Intelligent Optimization*, 2011. URL https://api.semanticscholar.org/CorpusID:6944647.

F. Hutter, H. H. Hoos, and K. Leyton-Brown. Parallel algorithm configuration. In *Learning and Intelligent Optimization: 6th International Conference, LION 6, Paris, France, January 16-20, 2012, Revised Selected Papers*, pages 55–70. Springer, 2012.

F. Hutter, H. Hoos, and K. Leyton-Brown. An efficient approach for assessing hyperparameter importance. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page I–754–I–762. JMLR.org, 2014a.

F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown. Algorithm runtime prediction: Methods and evaluation. *Artificial Intelligence*, 206:79–111, 2014b. ISSN 0004-3702. doi: https://doi.org/10.1016/j.artint.2013.10.003. URL https://www.sciencedirect.com/science/article/pii/S0004370213001082.

H. Ishibuchi, H. Masuda, Y. Tanigaki, and Y. Nojima. Modified distance calculation in generational distance and inverted generational distance. In *Evolutionary Multi-Criterion Optimization: 8th International Conference, EMO 2015, Guimarães, Portugal, March 29–April 1, 2015. Proceedings, Part II 8*, pages 110–125. Springer, 2015.

G. James, D. Witten, T. Hastie, R. Tibshirani, et al. *An introduction to statistical learning*, volume 112. Springer, 2013.

K. Jamieson and A. Talwalkar. Non-stochastic best arm identification and hyperparameter optimization. In *Artificial intelligence and statistics*, pages 240–248. PMLR, 2016.

D. R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of global optimization*, 21:345–383, 2001.

D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13:455–492, 1998.

M. I. Jordan and T. M. Mitchell. Machine learning: Trends, perspectives, and prospects. *Science*, 349(6245):255–260, 2015.

F. Karl, T. Pielok, J. Moosbauer, F. Pfisterer, S. Coors, M. Binder, L. Schneider, J. Thomas, J. Richter, M. Lang, E. C. Garrido-Merchán, J. Branke, and B. Bischl. Multi-objective hyperparameter optimization – an overview, 2022.

A. Klein and F. Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization. *CoRR*, abs/1905.04970, 2019a. URL http://arxiv.org/abs/1905.04970.

A. Klein and F. Hutter. Tabular benchmarks for joint architecture and hyperparameter optimization. *CoRR*, abs/1905.04970, 2019b. URL http://arxiv.org/abs/1905.04970.

A. Klein, S. Falkner, J. T. Springenberg, and F. Hutter. Learning curve prediction with bayesian neural networks. In *International conference on learning representations*, 2016.

A. Klein, S. Falkner, S. Bartels, P. Hennig, and F. Hutter. Fast bayesian optimization of machine learning hyperparameters on large datasets. In *Artificial intelligence and statistics*, pages 528–536. PMLR, 2017.

J. Knowles. Parego: a hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1):50–66, 2006. doi: 10.1109/TEVC.2005.851274.

L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, 18(25):1–5, 2017.

H. J. Kushner. A New Method of Locating the Maximum Point of an Arbitrary Multipeak Curve in the Presence of Noise. *Journal of Basic Engineering*, 86(1):97–106, 03 1964. ISSN 0021-9223. doi: 10.1115/1.3653121. URL https://doi.org/10.1115/1.3653121.

B. Lakshminarayanan. *Decision trees and forests: a probabilistic perspective*. PhD thesis, UCL (University College London), 2016.

B. Lakshminarayanan, D. M. Roy, and Y. W. Teh. Mondrian forests for large-scale regression when uncertainty matters. In *Artificial Intelligence and Statistics*, pages 1478–1487. PMLR, 2016.

B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. 2017. URL http://arxiv.org/abs/1612.01474.

J. Larson, M. Menickelly, and S. M. Wild. Derivative-free optimization methods. *Acta Numerica*, 28:287–404, 2019.

S. Le Digabel and S. M. Wild. A taxonomy of constraints in simulation-based optimization. Preprint ANL/MCS-P5350-0515, Argonne National Laboratory, Mathematics and Computer Science Division, 2015. URL http://www.mcs.anl.gov/papers/P5350-0515.pdf.

L. Li, M. Khodak, M.-F. Balcan, and A. Talwalkar. Geometry-aware gradient algorithms for neural architecture search. URL http://arxiv.org/abs/2004.07802.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *The journal of machine learning research*, 18(1):6765–6816, 2017.

L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018.

L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-Tzur, M. Hardt, B. Recht, and A. Talwalkar. A system for massively parallel hyperparameter tuning. *Proceedings of Machine Learning and Systems*, 2:230–246, 2020.

R. Liaw, E. Liang, R. Nishihara, P. Moritz, J. E. Gonzalez, and I. Stoica. Tune: A research platform for distributed model selection and training. *arXiv preprint arXiv:1807.05118*, 2018.

M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, and F. Hutter. Smac3: A versatile bayesian optimization package for hyperparameter optimization. *Journal of Machine Learning Research*, 23(54):1–9, 2022. URL http://jmlr.org/papers/v23/21-0888.html.

H. Liu, Y.-S. Ong, X. Shen, and J. Cai. When Gaussian Process Meets Big Data: A Review of Scalable GPs. *arXiv:1807.01065 [cs, stat]*, Apr. 2019a. URL http://arxiv.org/abs/1807.01065. arXiv: 1807.01065.

H. Liu, K. Simonyan, and Y. Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019b. URL https://openreview.net/forum?id=S1eYHoC5FX.

Y. Liu, R. Hu, A. Kraus, P. Balaprakash, and A. Obabko. Data-driven modeling of coarse mesh turbulence for reactor transient analysis using convolutional recurrent neural networks. *Nuclear Engineering and Design*, 390:111716, 2022.

Z. Liu. *Automated Deep Learning: Principles and Practice*. PhD thesis, Université Paris-Saclay, 2021.

Z. Liu, A. Pavao, Z. Xu, S. Escalera, F. Ferreira, I. Guyon, S. Hong, F. Hutter, R. Ji, J. C. J. Junior, et al. Winning solutions and post-challenge analyses of the chalearn autodl challenge 2019. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(9):3108–3125, 2021.

W. J. Maddox, P. Izmailov, T. Garipov, D. P. Vetrov, and A. G. Wilson. A simple baseline for Bayesian uncertainty in deep learning. *Advances in Neural Information Processing Systems*, 32:13153–13164, 2019.

S. Madireddy, A. Yanguas-Gil, and P. Balaprakash. Improving performance in continual learning tasks using bio-inspired architectures. In *Conference on Lifelong Learning Agents*, pages 992–1008. PMLR, 2023.

T. Mallick, P. Balaprakash, E. Rask, and J. Macfarlane. Graph-partitioning-based diffusion convolutional recurrent neural network for large-scale traffic forecasting. *Transportation Research Record*, 2674(9):473–488, 2020.

T. Mallick, P. Balaprakash, and J. Macfarlane. Deep-ensemble-based uncertainty quantification in spatiotemporal graph neural networks for traffic forecasting. *arXiv preprint arXiv:2204.01618*, 2022.

M. Märtens and D. Izzo. The asynchronous island model and nsga-ii: study of a new migration operator and its performance. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1173–1180, 2013.

R. Maulik, R. Égelé, B. Lusch, and P. Balaprakash. Recurrent neural network architecture search for geophysical emulation. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '20. IEEE Press, 2020. ISBN 9781728199986.

R. Maulik, B. Lusch, and P. Balaprakash. Reduced-order modeling of advection-dominated systems with recurrent neural networks and convolutional autoencoders. *Physics of Fluids*, 33(3):037106, 2021.

R. Maulik, R. Égelé, K. Raghavan, and P. Balaprakash. Quantifying uncertainty for deep learning based forecasting and flow-reconstruction using neural architecture search ensembles, Nov. 2023. URL http://dx.doi.org/10.1016/j.physd.2023.133852.

N. Mehrabi, F. Morstatter, N. Saxena, K. Lerman, and A. Galstyan. A survey on bias and fairness in machine learning. *ACM computing surveys (CSUR)*, 54(6):1–35, 2021.

J. Mockus, V. Tiesis, and A. Zilinskas. The application of bayesian methods for seeking the extremum. *Towards global optimization*, 2(117-129):2, 1978.

S. Mohamed and B. Lakshminarayanan. Learning in implicit generative models. *arXiv preprint:1610.03483*, 2016.

F. Mohr and J. N. van Rijn. Learning curves for decision making in supervised machine learning - A survey. *CoRR*, abs/2201.12150, 2022. URL https://arxiv.org/abs/2201.12150.

F. Mohr and J. N. van Rijn. Fast and informative model selection using learning curve cross-validation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2023.

F. Mohr, T. J. Viering, M. Loog, and J. N. van Rijn. Lcdb 1.0: An extensive learning curves database for classification tasks. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 3–19. Springer, 2022.

A. Muhammad and S.-H. Bae. A survey on efficient methods for adversarial robustness. *IEEE Access*, 10:118815–118830, 2022.

S. Müller, N. Hollmann, S. P. Arango, J. Grabocka, and F. Hutter. Transformers can do bayesian inference. *arXiv preprint arXiv:2112.10510*, 2021.

R. M. Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

D. Nix and A. Weigend. Estimating the mean and variance of the target probability distribution. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 1, pages 55–60 vol.1, 1994. doi: 10.1109/ICNN.1994.374138.

D. M. Olsson and L. S. Nelson. The nelder-mead simplex procedure for function minimization. *Technometrics*, 17(1):45–51, 1975.

Optuna developers. Optuna api: optuna.samplers. https://optuna.readthedocs.io/en/stable/reference/samplers/index.html, 2018. Accessed: Jun 27, 2023.

Y. Ovadia, E. Fertig, J. Ren, Z. Nado, D. Sculley, S. Nowozin, J. V. Dillon, B. Lakshminarayanan, and J. Snoek. Can you trust your model's uncertainty? evaluating predictive uncertainty under dataset shift. *arXiv preprint arXiv:1906.02530*, 2019.

Y. Ozaki, Y. Tanigaki, S. Watanabe, and M. Onishi. Multiobjective tree-structured parzen estimator for computationally expensive optimization problems. In *Proc. the 2020 Genetic and Evolutionary Computation Conference (GECCO '20)*, page 533–541, 2020. doi: 10.1145/3377930.3389817.

Y. Ozaki, Y. Tanigaki, S. Watanabe, M. Nomura, and M. Onishi. Multiobjective tree-structured parzen estimator. *Journal of Artificial Intelligence Research*, 73:1209–1250, 2022.

H. Park, R. Zhu, E. Huerta, S. Chaudhuri, E. Tajkhorshid, and D. Cooper. End-to-end ai framework for interpretable prediction of molecular and crystal properties. *Machine Learning: Science and Technology*, 4(2):025036, 2023.

A. Pavão. *Methodology for Design and Analysis of Machine Learning Competitions*. PhD thesis, Université Paris-Saclay, 2023.

T. Pearce, A. Brintrup, M. Zaki, and A. Neely. High-quality prediction intervals for deep learning: A distribution-free, ensembled approach. In *International Conference on Machine Learning*, pages 4075–4084. PMLR, 2018.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

F. Pfisterer, L. Schneider, J. Moosbauer, M. Binder, and B. Bischl. Yahpo gym-an efficient multi-objective multi-fidelity benchmark for hyperparameter optimization. In *International Conference on Automated Machine Learning*, pages 3–1. PMLR, 2022.

H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean. Efficient neural architecture search via parameters sharing. In *International conference on machine learning*, pages 4095–4104. PMLR, 2018.

I. Pinelis. Order statistics on the spacings between order statistics for the uniform distribution. *arXiv preprint arXiv:1909.06406*, 2019.

R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm intelligence*, 1(1): 33–57, 2007.

R. Pugliese, S. Regondi, and R. Marini. Machine learning-based approach: global trends, research directions, and regulatory standpoints. *Data Science and Management*, 4:19–29, 2021. ISSN 2666-7649. doi: https://doi.org/10.1016/j.dsm.2021.12.002. URL https://www.sciencedirect.com/science/article/pii/S2666764921000485.

C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. Adaptive computation and machine learning. MIT Press, 2006. ISBN 026218253X.

E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. Regularized evolution for image classifier architecture search. 2018. URL http://arxiv.org/abs/1802.01548.

B. Recht, R. Roelofs, L. Schmidt, and V. Shankar. Do imagenet classifiers generalize to imagenet? In *International conference on machine learning*, pages 5389–5400. PMLR, 2019.

M. R. Rudary. *On predictive linear Gaussian models*. University of Michigan, 2009.

T. Ruhkopf, A. Mohan, D. Deng, A. Tornede, F. Hutter, and M. Lindauer. Masif: Meta-learned algorithm selection using implicit fidelity information. *Transactions on Machine Learning Research*, 2022.

R. L. Russell and C. Reale. Multivariate uncertainty in deep learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

R. A. Rutenbar. Simulated annealing algorithms: An overview. *IEEE Circuits and Devices magazine*, 5(1):19–26, 1989.

J. Sevilla, L. Heim, A. Ho, T. Besiroglu, M. Hobbhahn, and P. Villalobos. Compute trends across three eras of machine learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.

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. *Proceedings of the IEEE*, 104(1): 148–175, Jan. 2016. ISSN 0018-9219, 1558-2256. doi: 10.1109/JPROC.2015.2494218. URL https://ieeexplore.ieee.org/document/7352306/.

M. Sipser. Introduction to the theory of computation. *ACM Sigact News*, 27(1):27–29, 1996.

J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. *Advances in neural information processing systems*, 25, 2012.

J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pages 2171–2180. PMLR, 2015.

H. Song, M. Perello-Nieto, R. Santos-Rodriguez, M. Kull, P. Flach, et al. Classifier calibration: A survey on how to assess and improve predicted class probabilities. *arXiv preprint arXiv:2112.10327*, 2021.

N. Srinivas, A. Krause, S. Kakade, and M. Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 1015–1022, Madison, WI, USA, 2010. Omnipress. ISBN 9781605589077.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning research*, 15(1):1929–1958, 2014.

Y. Sun, T. Mallick, P. Balaprakash, and J. Macfarlane. A data-centric weak supervised learning for highway traffic incident detection. *Accident Analysis & Prevention*, 176:106779, 2022.

M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, and Q. V. Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2820–2828, 2019.

C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855, 2013.

E. Tjoa and C. Guan. A survey on explainable artificial intelligence (xai): Toward medical xai. *IEEE transactions on neural networks and learning systems*, 32(11):4793–4813, 2020.

J. van Amersfoort, L. Smith, A. Jesson, O. Key, and Y. Gal. On feature collapse and deep kernel learning for single forward pass uncertainty. *arXiv preprint arXiv:2102.11409*, 2021.

V. Vapnik. Principles of risk minimization for learning theory. *Advances in neural information processing systems*, 4, 1991.

T. Viering and M. Loog. The shape of learning curves: a review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Z. Wang and S. Jegelka. Max-value entropy search for efficient bayesian optimization. In *International Conference on Machine Learning*, pages 3627–3635. PMLR, 2017.

Z. Wang, C. Gehring, P. Kohli, and S. Jegelka. Batched large-scale bayesian optimization in high-dimensional spaces. In *International Conference on Artificial Intelligence and Statistics*, pages 745–754. PMLR, 2018.

F. Wenzel, J. Snoek, D. Tran, and R. Jenatton. Hyperparameter ensembles for robustness and uncertainty quantification. *Advances in Neural Information Processing Systems*, 33: 6514–6527, 2020.

F. Wenzel, J. Snoek, D. Tran, and R. Jenatton. Hyperparameter ensembles for robustness and uncertainty quantification. 2021. URL http://arxiv.org/abs/2006.13570.

S. M. Wild, J. Sarich, and N. Schunck. Derivative-free optimization for parameter estimation in computational nuclear physics. *Journal of Physics G: Nuclear and Particle Physics*, 42 (3):034031, feb 2015. doi: 10.1088/0954-3899/42/3/034031. URL https://doi.org/10.1088%2F0954-3899%2F42%2F3%2F034031.

A. G. Wilson and P. Izmailov. Bayesian deep learning and a probabilistic perspective of generalization. *arXiv preprint arXiv:2002.08791*, 2020.

J. M. Wozniak, R. Jain, P. Balaprakash, J. Ozik, N. T. Collier, J. Bauer, F. Xia, T. Brettin, R. Stevens, J. Mohd-Yusof, et al. Candle/supervisor: A workflow framework for machine learning applied to cancer research. *BMC bioinformatics*, 19:59–69, 2018.

F. Xia, M. Shukla, T. Brettin, C. Garcia-Cardona, J. Cohn, J. E. Allen, S. Maslov, S. L. Holbeck, J. H. Doroshow, Y. A. Evrard, E. A. Stahlberg, and R. L. Stevens. Predicting tumor cell line response to drug pairs with deep learning. *BMC Bioinformatics*, 19(S18):486, Dec. 2018. ISSN 1471-2105. doi: 10.1186/s12859-018-2509-3. URL https://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-018-2509-3.

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

X. Ying. An overview of overfitting and its solutions. In *Journal of physics: Conference series*, volume 1168, page 022022. IOP Publishing, 2019.

T. Yu and H. Zhu. Hyper-parameter optimization: A review of algorithms and applications. *arXiv preprint arXiv:2003.05689*, 2020.

S. Zaidi, A. Zela, T. Elsken, C. Holmes, F. Hutter, and Y. W. Teh. Neural ensemble search for uncertainty estimation and dataset shift. *arXiv preprint arXiv:2006.08573*, 2020.

A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter. Understanding and robustifying differentiable architecture search. URL http://arxiv.org/abs/1909.09656.

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.

R. Égelé, R. Maulik, K. Raghavan, B. Lusch, I. Guyon, and P. Balaprakash. AutoDEUQ: Automated deep ensemble with uncertainty quantification. In *26th International Conference on Pattern Recognition (ICPR)*, pages 1908–1914. IEEE, 2022.

R. Égelé, T. Chang, Y. Sun, V. Vishwanath, and P. Balaprakash. Parallel multi-objective hyperparameter optimization with uniform normalization and bounded objectives. *arXiv preprint arXiv:2309.14936*, 2023a.

R. Égelé, I. Guyon, Y. Sun, and P. Balaprakash. Is one epoch all you need for multi-fidelity hyperparameter optimization? *ESANN 2023 proceedings*, 2023b. doi: 10.14428/esann/2023.es2023-84. URL http://dx.doi.org/10.14428/esann/2023.ES2023-84.