Future Generation Computer Systems 175 (2026) 108042

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Resource-adaptive successive doubling for hyperparameter optimization with large datasets on high-performance computing systems

Marcel Aach [a,b,*], Rakesh Sarma [a], Helmut Neukirchen [b], Morris Riedel [a,b], Andreas Lintermann [a]

[a] Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Germany
[b] School of Engineering and Natural Sciences, University of Iceland, Iceland

ARTICLE INFO

Keywords:
Hyperparameter optimization
High-performance computing
Distributed deep learning
Machine learning

ABSTRACT

The accuracy of Machine Learning (ML) models is highly dependent on the hyperparameters that have to be chosen by the user before the training. However, finding the optimal set of hyperparameters is a complex process, as many different parameter combinations need to be evaluated, and obtaining the accuracy of each combination usually requires a full training run. It is therefore of great interest to reduce the computational runtime of this process. On High-Performance Computing (HPC) systems, several configurations can be evaluated in parallel to speed up this Hyperparameter Optimization (HPO). State-of-the-art HPO methods follow a bandit-based approach and build on top of successive halving, where the final performance of a combination is estimated based on a lower than fully trained fidelity performance metric and more promising combinations are assigned more resources over time. Frequently, the number of epochs is treated as a resource, letting more promising combinations train longer. Another option is to use the number of workers as a resource and directly allocate more workers to more promising configurations via data-parallel training. This article proposes a novel Resource-Adaptive Successive Doubling Algorithm (RASDA), which combines a resource-adaptive successive doubling scheme with the plain Asynchronous Successive Halving Algorithm (ASHA). Scalability of this approach is shown on up to 1,024 Graphics Processing Units (GPUs) on modern HPC systems. It is applied to different types of Neural Networks (NNs) and trained on large datasets from the Computer Vision (CV), Computational Fluid Dynamics (CFD), and Additive Manufacturing (AM) domains, where performing more than one full training run is usually infeasible. Empirical results show that RASDA outperforms ASHA by a factor of up to 1.9 with respect to the runtime. At the same time, the solution quality of final ASHA models is maintained or even surpassed by the implicit batch size scheduling of RASDA. With RASDA, systematic HPO is applied to a terabyte-scale scientific dataset for the first time in the literature, enabling efficient optimization of complex models on massive scientific data.

1. Introduction

In recent years, the amount of openly available data has drastically increased. This includes datasets from different scientific fields, such as CV [1], Earth Observation (EO) [2], High-Energy Physics (HEP) [3], AM [4], or CFD [5]. To analyze these data efficiently and gain novel insights based on hidden correlations, the use of Deep Learning (DL) techniques and NNs has become essential due to their ability to automatically extract complex patterns. As the prediction quality of these NN models is highly dependent on the so-called hyperparameters, which are frequently related to, e.g., the NN architecture or the optimizer, systematic HPO has become a crucial ingredient of ML workflows [6]. However, this search for optimal combinations of hyperparameters is challenging due to often high-dimensional search spaces. Furthermore, the performance of a sample from the search space can only be evaluated with a high degree of confidence after a full model training run. In the case of deep NNs trained on large datasets, this can become a major hurdle, even with extensive computing resources. Additionally, the search space is often diverse in nature. For example, the search space could be comprised of the learning rate, an optimizer-related parameter represented as floating point number, and the number of layers, an architectural parameter represented as an integer number $l > 0$. Categorical values, such as "type of optimizer" or "type of layer" are also possible. This makes the application of classical, gradient-based optimization methods infeasible. Hyperparameters also change under different models and datasets, making the generalization difficult to

* Corresponding author at: Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, Germany.
E-mail address: m.aach@fz-juelich.de (M. Aach).

https://doi.org/10.1016/j.future.2025.108042
Received 10 December 2024; Received in revised form 15 July 2025; Accepted 23 July 2025
Available online 7 August 2025

assess. One of the state-of-the-art HPO methods is the ASHA [7]. It randomly samples multiple combinations, evaluates their performance with a lower training budget and then – after comparing their performance – terminates under-performing trials early on. To reduce the time to solution, ASHA is frequently executed in parallel, where multiple NN configurations (trials) are evaluated at the same time.

Modern HPC systems offer a natural setting for running this kind of workload. They feature accelerators, such as GPUs, that are ideally suited for efficient NN trainings (*fast computation*). Furthermore, these accelerators are connected by an optimized communication network that enables *fast inter-node communication*. While current distributed HPO methods, such as ASHA, leverage the fast computation capabilities to train different hyperparameter candidates, the communication requirements are usually modest and limited to the exchange of the value of a certain metric, e.g., the current loss on the validation set for the comparison of the performance between trials.

This work introduces a novel method, the RASDA, that leverages *both* HPC features to perform HPO efficiently at scale. It combines two levels of parallelism: (i) on the HPO level, different trials are run in parallel and (ii) on the level of each trial run, the NN training is accelerated with data-parallel training. The latter splits the datasets onto multiple GPUs and performs gradient synchronization after each training step. As these gradients are typically large, they require high-bandwidth communication. RASDA then leverages the successive doubling principle, which progressively allocates more resources to more promising hyperparameter combinations, treating the amount of GPUs that are used for data-parallel training as resources (performing a doubling in space). In contrast, other successive halving techniques, such as the plain ASHA, treat the number of epochs during training of a model as resources and thus perform only halving in time, see Fig. 1.

The developed method is suitable for problems that involve large scientific datasets, where due to long training times, even with HPC resources it is not feasible to train more than the initially sampled hyperparameter configurations and users are interested in getting the best possible, fully-trained model in the shortest amount of time. Therefore, this study performs an extensive evaluation of RASDA on different datasets from the CV, CFD, and AM domains, which are up to 8.3 Terrabyte (TB) in size, to prove its capability to deal with large datasets. These datasets are used to tune the hyperparameters of different types of NNs, namely a Convolutional Neural Network (CNN), an autoencoder and a transformer. RASDA is also benchmarked against the current state-of-the-art successive halving HPO method ASHA. The new RASDA code is openly available on GitHub[1] for the community, see Table A.8 for an overview of the repository.

In summary, the key contributions of RASDA are:

- Combination of successive halving in space with successive doubling in time, allocating more GPUs to more promising trials.
- Reduces the runtime of more promising hyperparameter trials by leveraging a higher degree of parallelism in data-parallel training
- Leverages the inherent features of HPC systems, fast computation for the training, and fast communication for exchange of gradients during distributed training
- Outperforms the plain ASHA method in runtime and model performance across different domains and on datasets up to 8.3 TB.

This article is structured as follows. Section 2 summarizes the related work and highlights the differences to this work. The main details of RASDA are presented in Section 3. The application cases are explained in Section 4, followed by a presentation and discussion of the empirical results of the algorithm in Section 5. Finally, a summary and outlook are provided in Section 6.

[1] RASDA source: https://github.com/olympiquemarcel/rasda.

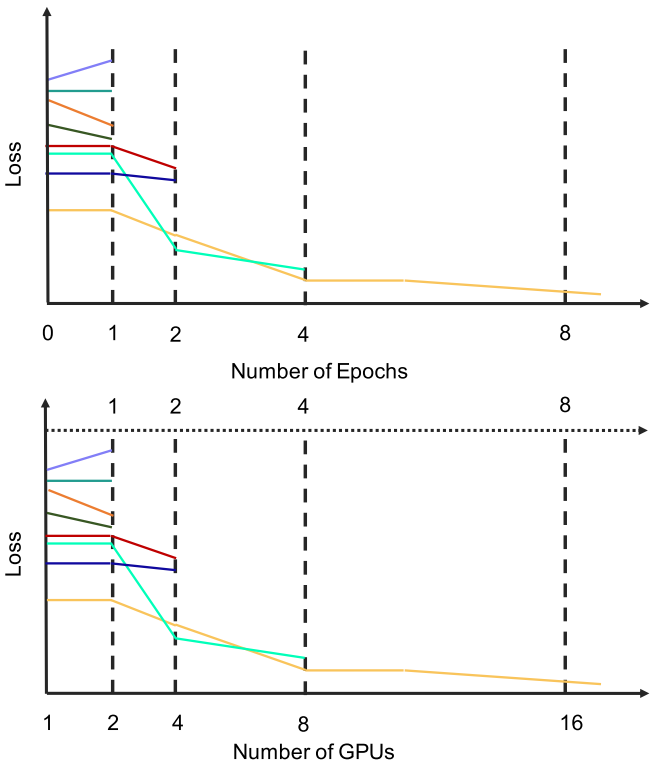


Fig. 1. Comparison of successive halving in time (top) and halving in time combined with doubling in space (bottom). Each line corresponds to the learning curve of a single HPO combination.

2. Related work

In ML, the performance of a certain model measured by a specific metric, such as the validation error, can be represented by the function $f : \mathcal{X} \to \mathbb{R}$ where $\mathcal{X}$ denotes the space of possible hyperparameter combinations. The primary goal of HPO is to minimize the objective function f by identifying a hyperparameter configuration $x^* \in \mathcal{X}$ such that $x^* \in \arg\min_{x \in \mathcal{X}} f(x)$. Evaluations of the objective function are costly because they typically involve fully training the model for each configuration. To optimize this workflow, several approaches exist. These are either based on approximating $f(x)$ by a lower fidelity estimate, e.g., by the performance after a few training epochs or a model trained on a fraction of the data, or on choosing better hyperparameter configurations to evaluate, e.g., using BO. This section summarizes these approaches, i.e. Section 2.1 describes the successive halving method, Sections 2.2 and 2.3 summarize resource-adaptive as well as other HPO algorithms and Section 2.4 introduces the concept of data-parallel training.

2.1. Successive halving

Successive halving is a variant of Random Search [8], which uses the fact that most ML algorithms are iterative in nature. Intermediate performance results are thus accessible long before the algorithm is fully trained. The problem of finding optimal hyperparameters in a vast search space can then be framed in the context of a multi-armed bandit problem, where each arm represents a hyperparameter combination, and pulling an arm corresponds to training the combination for some iterations [9]. The goal is to identify the arm that yields the highest reward with the lowest budget possible. To do so in an efficient way, successive halving uniformly allocates an initial budget B to n_{arms} arms and evaluates their performance after a few iterations at a milestone with budget B/n_{arms}. It then eliminates the worst-performing half of the

arms and promotes the most promising-performing arms by continuing to pull them. Each of these successive halving steps is referred to as a rung. When following this procedure for a few steps from rung to rung, only one arm, i.e., the one with the best performance, remains at the end. Hyperband (HB) [10] extends this concept by iterating over different numbers of initial arms n_{arms} (also referred to as brackets) to evaluate.

However, when performing HPO at a larger scale, these methods are sensitive to so-called stragglers. To determine the combinations belonging to the under- and top-performing half, the performance measurement for all combinations needs to be available, which means that faster trials need to wait for the slower ones. ASHA addresses this scalability problem by deciding on a rolling basis which trials are worth continuing. When two trials have finished their initial number of iterations, the trial with the better performance is promoted. At the same time, the other trial is paused until the performance of the next completed trial can be juxtaposed. In contrast to HB, ASHA is mostly performed with only a single bracket, and was evaluated on up to 500 GPUs in [7].

Another possibility of finding a minimum of the objective function f is to use black-box optimization methods such as BO. The idea behind BO is to use a probabilistic model of f that is based on data points observed in the past. In the case of HPO, this corresponds to finding new promising hyperparameter combinations based on the performance of past combinations. The BOHB algorithm [11] combines the BO process with HB for scheduling. To this aim, HB is used to choose the number of hyperparameter configurations and their assigned budget, while BO is used to choose the hyperparameters by deploying a tree parzen estimator [12].

The mentioned methods have in common that they focus only on identifying the most promising arm and delivering that hyperparameter combination as a result at the end of a run. In contrast, RASDA also ensures the full training of the best combination to yield a complete model.

2.2. Resource-adaptive schedulers

Most of the existing successive halving-based HPO schedulers treat the number of epochs or training time as resources (also known as fidelity in the literature). It is, however, also possible to treat the spatial amount of computational resources, e.g., the number of GPU, used for training a model as a fidelity. A low-fidelity measurement then corresponds to the performance of a NN trained with a small number of devices. The most relevant existing HPO schedulers that focus on this computational resource-adaptive scheduling are presented in the following.

HyperSched [13] introduces a scheduler to dynamically allocate resources in time and space to the best-performing hyperparameter trials. It thereby not only identifies the most promising model but also trains it – ideally fully – by a fixed deadline. The main novelty of the algorithm is its deadline awareness, which means that it schedules fewer new trials as it approaches the deadline. This way, the exploration of new configurations is stopped in favor of deeper exploitation of the running trials. HyperSched is evaluated in [13] on different CV benchmarking datasets on up to 32 GPUs on Cloud computing instances.

Rubberband [14] extends HyperSched by leveraging the elasticity of the Cloud for the task of scheduling HPO workloads. It takes into account not only the performance of a combination but also the financial costs of a GPU hour, with the goal of minimizing the costs of an HPO job. Based on the idea of diminishing returns when scaling the training of a single model, the algorithm de-allocates resources (and thus saves costs) from less-promising trials, once a promising trial has been identified. It also creates a resource allocation plan a priori the run to optimize the performance of the single trials that are trained via distributed DL. The resource allocation plan is initialized with an initial burn-in period during which training latencies and scaling performance of trials are measured.

Sequential Elimination with Elastic Resources (SEER) [15] further takes advantage of the elasticity in the cloud by adaptively allocating and de-allocating compute resources during the HPO run. At the same time, it focuses on maximizing the accuracy of trials, in combination with minimizing the total financial cost. Therefore, it limits the amount of workers allocated to the top trials once sub-linear scaling performance sets in.

Both Rubberband and SEER rely heavily on the adaptive allocation and de-allocation of GPU instances, which is possible in an elastic cloud setting but not on HPC systems, where the amount of GPUs allocated to the overall HPO job is usually static. HyperSched, meanwhile, focuses on maximizing the performance by the deadline. In contrast, the proposed RASDA method aims to deliver the best-possible result in the shortest amount of time.

2.3. Other HPO algorithms and libraries

Many other algorithms and libraries for performing HPO exist. These include BO-based libraries such as Dragonfly [16] and SMAC [17], allowing the user to select different surrogate models and acquisition functions. Optuna [18] also relies on BO and provides automated tracking and visualization of trials. Since parallel computing resources have become increasingly available in recent years, several algorithms have emphasized large-scale, distributed HPO: DeepHyper [19] focuses on performing asynchronous BO on HPC systems and has been applied to several scientific use cases [20–22]. Distributed evolutionary optimization can be performed with Propulate [23] and Population Based Training (PBT) [24].

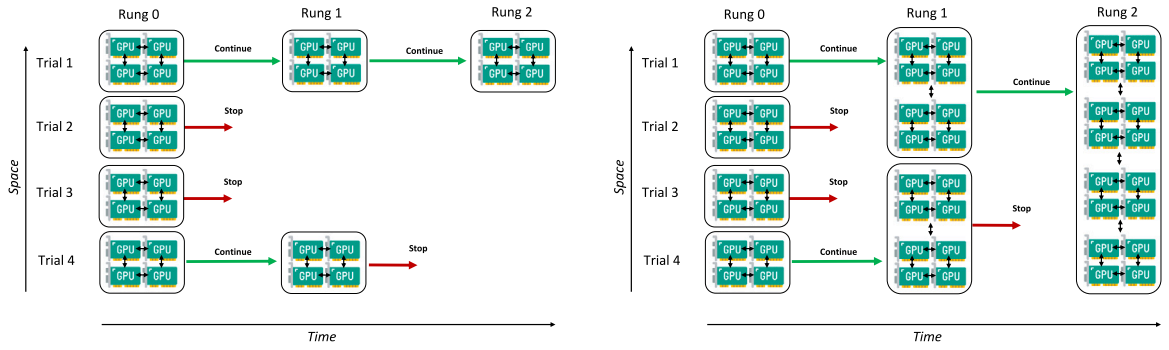
While most of these libraries support multi-fidelity HPO, none of them so far supports performing resource-adaptive scheduling of trials, which is, however, supported by RASDA.

2.4. Data-parallel deep learning

Data-parallel training is a technique to reduce the runtime of the training of DL models on large datasets by using multiple devices, such as GPUs. In data-parallel training, the training dataset $\mathcal{D}$ is divided among the number of workers N, where each worker is assigned an identical copy of the model to train on a distinct subset of the data $\mathcal{D}_1 \cup \mathcal{D}_2 \cup \cdots \cup \mathcal{D}_N$. Specifically, each worker $i = 1 \dots N$ runs one model forward and backward pass with a predefined number of samples, the local batch size BS_{local}, of its subset of data to compute its local gradients Δw_i with respect to the model parameters w. After the backward pass, these local gradients are aggregated and averaged across all workers by

$$\Delta w = \frac{1}{N} \sum_{i=1}^{N} \Delta w_i, \tag{1}$$

The averaged global gradient is then used to update the model parameters on all workers every $BS_{global} = BS_{local} \cdot N$ samples [25]. To remain computationally efficient, each worker needs a sufficient amount of data to run the training, thus BS_{local} needs to be large. At the same time, BS_{global} increases linearly with N. When BS_{global} becomes too large, it can impact the generalization performance for two reasons. First, the number of optimizer updates per epoch decreases, as an update is performed every BS_{global} samples. This can be addressed to some extent by scaling the learning rate with the number of devices [26]. This approach is, however, infeasible for an extremely large BS_{global}, since in such a case also the learning rate becomes too large. Second, Stochastic Gradient Descent (SGD) with large batch sizes tends to converge to *sharp minima* [27] which does not generalize well, see [28] for more details.



(a) ASHA (b) RASDA

Fig. 2. Comparison of (plain) ASHA, performing successive halving only in the time domain, and RASDA, performing successive halving in the time and successive doubling in the space domain at the same time on an GPU cluster. In the RASDA case, when a trial is terminated, its workers are allocated to the more promising trials to increase the parallelism of the data-parallel training. Black arrows indicate communication of gradients between GPUs.

Algorithm 1 Resource Adaptive Successive Doubling Method

Input: `trial_result, base_resources, sf, milestones`

```
if trial_result["training_iteration"] ∈ milestones then
    current_rung ← milestones.index(trial_result["training_iteration"])
    new_resources ← base_resources × sf^current_rung
    return new_resources
else
    return None
end if
```

3. Resource-adaptive successive doubling algorithm

This section presents details on RASDA in Section 3.1 and provides an explanation on how issues with large batch size training, cf. Section 2.4, are addressed in Section 3.2. The performance optimizations are presented in Section 3.3, while in Section 3.4 and Section 3.5 the compatibility of RASDA with other tools and its dependcy on the hardware setup are summarized.

3.1. Algorithm design and implementation

The main idea of RASDA is to combine a successive halving step in the time domain, i.e., train more promising configurations for longer, and a successive doubling step in the spatial domain, i.e., allocate more workers to more promising configurations. This way, when reaching a rung milestone, the worst-performing trials are terminated (halving in time) and the free workers are allocated to the top-performing trials (doubling in space), see Fig. 2. The additional workers are then used to increase the parallelism of the data-parallel training of the configuration, which leads to faster training times.

For the re-allocation of workers, a second successive doubling routine in addition to the successive halving routine of ASHA is used (the resource allocation part is described in Alg. 1): All trials start out with an initial number of workers (`base_resources`). When a trial reports a new `trial_result`, it is first checked if the current `training_iteration`, e.g., the current epoch, corresponds to one of the rung `milestones`. At every rung `milestone`, the (plain) ASHA scheduler then reduces the number of running trials by the reduction factor `rf`. The resources for all trials that are allowed to continue are then increased with the scaling factor `sf` by the RASDA scheduler, yielding the `new_resources` for the trial (following Alg. 1). If the reduction and scaling factors are equal, i.e. `rf = sf`, all workers are continuously allocated to a trial. In practice, however, some trials do run faster than others. The advantage of the ASHA and RASDA scheduler is that they both perform asynchronous halving and doubling, i.e., top-performing trials are promoted to the next rung even if not all trials in the current rung have reached their milestones. This reduces idling times between halving steps. It should be noted that due to this asynchronous execution, the percentage of trials terminated at each milestone can be smaller than `rf`. As the total number of workers in the system is a constant, the trials that are allowed to continue might need to wait until their new resource requirements are met.

At these rung `milestones`, two processes occur: the (plain) ASHA scheduler reduces the number of running trials by the reduction factor `rf`, while Alg. 1 handles the reallocation of GPU resources among the remaining trials.

The total number of rungs and their corresponding milestones in the RASDA scheduler are calculated based on the minimum and maximum iterations $\min\_t$ and $\max\_t$, along with the scaling factor `sf`, as

$$\texttt{num\_rungs} = \left\lfloor \frac{\log\left(\frac{\max\_t}{\min\_t}\right)}{\log(\texttt{sf})} \right\rfloor, \tag{2}$$

$$\texttt{rung\_milestones} = \min\_t \cdot \texttt{sf}^k, \tag{3}$$

with $k = 0, \ldots, \texttt{num\_rungs}$. This ensures a geometric progression of the milestones, as described for ASHA by Li et al. [7].

The algorithm is implemented with Ray Tune [29], an open-source library for performing distributed HPO. Ray Tune orchestrates the optimization process by launching a single head node and several worker nodes on an HPC cluster. The head node then connects to the worker nodes and starts the trials. During training, the worker nodes report their current status including performance metrics to the head node that makes scheduling decisions, such as termination or continuation of new trials.

Ray Tune already features implementations of several successive halving methods. The implementation of RASDA therefore relies on the implementation of ASHA that exists already inside of Ray Tune

for performing the time-wise successive halving. For the spatial successive doubling, RASDA makes use of the *ResourceChangingScheduler* interface,[2] enabling the modification of resource requirements for trials. At each milestone, the trial is saved, including the current weights of the model. If the decision is made to continue the training, the trial is relaunched with the new resource requirements. It should be noted, that the ASHA implementation of Ray Tune has some minor differences to the original algorithm in [7]. However, empirical evidence shows that these differences do not impact performance.

The data-parallel training part is handled by the PyTorch-DDP library [25], which uses the NVIDIA Collective Communications Library (NCCL) backend[3] for communication and gradient synchronization.

3.2. Large batch training

Recall from Section 2.4 that scaling the data-parallel training to a large number of devices and increasing BS_{global} can impact the generalization performance of models. The following provides an intuitive explanation of how this issue is addressed by the RASDA scheduler.

McCandlish et al. [30] empirically studied large batch training for various models: they introduce the Gradient Noise Scale (GNS) metric, which serves as a noise-to-signal measure of the training progress. In theory, if the true gradient G_{true} from performing full-batch Gradient Descent without the stochastic component would be available, it would be possible to compute a simple version of the GNS by

$$\mathrm{GNS}_{simple} = \frac{\mathrm{tr}(\Sigma)}{|G_{true}|^2}, \tag{4}$$

where Σ is the per-data-sample covariance matrix of G_{true}. Essentially, the nominator measures the noise of the gradient, while the denominator measures its magnitude. As the DL model converges, the gradient decreases in size, which results in an increase of the GNS over training time. McCandlish et al. use an approximation to compute the GNS based on the estimated stochastic gradient G_{est} and confirm that the GNS indeed increases over time.

Based on the GNS, Qiao et al. [31] introduce the concept of "statistical efficiency" of the DL training, measuring the amount of training progress made per data sample processed in a batch. The key insight is that when the GNS is low, there is no benefit for the learning progress in adding more data samples to the batch (thus increasing BS_{global}), as the stochastic gradient G_{est} is a precise approximation of G_{true} already. However, when the GNS is high, adding more data samples to the batch reduces the noise and leads to a better gradient approximation. As the GNS starts out small and increases over time, this justifies the usage of larger batch sizes during the later part of training. This approach has also been used successfully for HPO and scheduling tasks in the past [31,32].

Additionally, Smith et al. [33] find that increasing BS_{global} over time has a similar effect as decaying in the learning rate, which is common practice in DL nowadays [34]. Based on these findings, the following two insights can be derived:

- Training with a small BS_{global} generally helps generalization and is computationally efficient at the beginning of training, in terms of the training progress per processed data sample.
- Increasing BS_{global} over time and using a large BS_{global} as the model is converging is computationally efficient as well.

This aligns well with the scheduling of the RASDA algorithm. In the beginning, the trials train with a small BS_{global}, i.e., the number of workers allocated for the data-parallel training is small. As time progresses, BS_{global} increases with each resource doubling step, as more and more workers are allocated to the data-parallel training. The evaluation in Section 5 shows that by leveraging this approach, the generalization capabilities of the final models match or exceed those of models that are continuously trained with a small BS_{global}.

Another crucial point is the correct scaling of the learning rate with the batch size. In the evaluation in Section 5, the learning rate is scaled linearly with the number of workers, i.e., up to a factor of 8×, when using SGD [35]. Furthermore, it follows a square-root scaling rule when using Adaptive Moment Estimation (ADAM) [36]. In the case of re-scaling, the learning rate is not immediately scaled to a larger value. Instead, there is a warm-up over one or two epochs. This re-scaling parameter is included as a hyperparameter in the search space, see Table 1. Thereby, the HPO run automatically optimizes towards learning stability.

3.3. Performance optimization

To ensure efficient performance, several additional optimizations are made to the trials in the HPO loop. This includes selecting the BS_{local} sufficiently large such that it fills the GPU memory in addition to the model for each of the applications. As the training datasets have to be loaded by each trial in parallel when performing HPO, they are loaded into shared memory when they fit in size. Training datasets that do not fit into shared memory are stored on a partition of the file system with high bandwidth to avoid bottlenecks. For data loading, the native PyTorch data loader as well as the NVIDIA DALI library[4] are used.

A preliminary study determined that saving the model weights into a checkpoint too often can lead to bottlenecks [37]. Therefore, the checkpoint frequency is reduced to every five epochs and the rung milestones of the ASHA and RASDA scheduler are adjusted accordingly. Ray Tune needs an initial start-up time to launch the head node and all connected worker nodes. As this is the same for ASHA and RASDA, these timings are excluded from the measurements.

3.4. Compatibility with existing HPO tools

As RASDA functions as a pure scheduling tool, it can be integrated with various HPO and AutoML frameworks. Since it is already incorporated into the Ray Tune framework via the scheduler interface, it can be seamlessly used within workflows that leverage the scheduling infrastructure of Ray Tune. This includes BO tools such as BOHB [11], Optuna [18], and BayesOpt [38], as well as evolutionary optimization frameworks like HEBO [39]. In such cases, the Bayesian or evolutionary algorithm proposes new hyperparameter candidates to sample from the search space, while RASDAs time-wise successive halving determines which trials to terminate at different points in time. Concurrently, its space-wise successive doubling allocates additional GPU resources to the trials selected to continue.

3.5. Dependency on hardware setup

As the main idea of RASDA is to perform successive doubling in the spatial domain, the scheduler operates most efficiently on HPC systems where the number of GPUs per node, the total number of allocated nodes, and the number of concurrent trials follow a power-of-two configuration. This setup ensures that GPUs resources can be reassigned seamlessly across rung milestones, as illustrated in Fig. 2. However, such configurations may not always be available in practice. In scenarios where the number of GPUs is not a power of two or where the number of trials exceeds the number of available GPUs, RASDA relies on the underlying Ray Tune framework to manage resource scheduling and queuing. When more trials are submitted than there are GPUs available, not all trials can be launched simultaneously. In this

[2] ResourceChangingScheduler (version 2.8.0): https://docs.ray.io/en/latest/tune/api/doc/ray.tune.schedulers.ResourceChangingScheduler.html.

[3] NCCL backend: https://github.com/NVIDIA/nccl.

[4] DALI: https://developer.nvidia.com/dali.

Table 1
Search space for the experiments, comprised of several optimizer-related and architectural parameters. Superscripts indicate which hyperparameters are used as search space for which applications: CV, CFD, AM. The "re-scaling warm-up" parameter handles the gradual increase of the learning rate when the number of devices and with it BS_{global} is increased.

Hyperparameter	Type	Range
Learning rate$^{CV,\ CFD,\ AM}$	Float	log[1e−5, 1]
Weight decay$^{CV,\ CFD,\ AM}$	Float	log(0, 1e−1]
Initial warm-up$^{CV,\ CFD,\ AM}$	Int	[1,2,3,4,5]
Optimizer$^{CV,\ CFD,\ AM}$	Cat	["sgd", "adam"]
Layer initialization$^{CV,\ CFD}$	Cat	["kaiming" [[40]], "xavier"[41]]
Activation function$^{CV,\ CFD}$	Cat	["ReLU", "LeakyReLU", "SELU", "Tanh", "Sigmoid"]
Convolution kernel size$^{CV,\ CFD}$	Int	[5,7,9]
Re-scaling warm-up$^{CFD,\ AM}$	Int	[1,2]
Patch size^{AM}	Int	[2, 4]
Depth^{AM}	Int	[1, 2, 4]
Number of attention heads^{AM}	Int	[3, 6, 12, 24]
MLP ratio^{AM}	Float	[1., 2., 3., 4.]

case, RASDA proceeds with its successive doubling routine for the first batch of trials that are scheduled. Remaining trials are queued and executed as resources become available. At the milestones, trials selected to continue are paused if their requested GPU allocation cannot be satisfied immediately, and they are resumed once sufficient resources are freed by completed or terminated trials. Although this queuing may introduce some latency, the promoted trials benefit from increased parallelism once resumed, resulting in significantly faster training. The benefit becomes especially pronounced in higher rungs, where larger resource allocations substantially reduce the training time per epoch. As a result, RASDA is still expected to deliver well-performing hyperparameter candidates faster than plain ASHA. Similarly, in cases where the total number of GPUs is not a power of two, the successive doubling scheme can still be applied, although some adaptation is required. Specifically, the values of `sf`, `rf`, min_t, and max_t should be chosen to ensure that the maximum number of GPUs allocated per trial in the final rung does not exceed the available resources. It is generally advisable to avoid resource fragmentation across nodes, as splitting the GPUs on a node between multiple trials may reduce the efficiency of data-parallel training.

4. Application cases

To assess the proposed RASDA scheduler, its performance is evaluated across a range of different tasks from the CV, CFD, and AM domain on seperate training, validation and test dataset splits to avoid overfitting. The different cases feature various models with different hyperparameters to optimize as well as training datasets of different sizes. The application domains and the set-up of these tasks is described in the following.

4.1. Computer vision

For the CV domain, the hyperparameters of a ResNet50 [42] trained on the ImageNet dataset [1] are optimized, as this is still one of the most important reference benchmarks [43]. The ImageNet dataset contains 1,281,167 training images and 50,000 validation images divided into 1000 object classes. In TFRecord file format, the dataset is approximately 146 Gigabyte (GB) in size.

The ResNet follows a basic CNN architecture with multiple residual connections between layers. The HPO search space for the ResNet includes several architectural hyperparameters, e.g., the type of activation functions or size of the input convolution kernel, as well as optimizer-related parameters, such as the learning rate or weight decay, see Table 1 for an exhaustive list.

All models are trained for min_t = 5 to max_t = 40 epochs and a reduction and scaling factor `sf` = `rf` = 2 is chosen for the schedulers. Following Eq. (3), this results in rung milestones at epochs 5, 10, 20, and 40.

As classification accuracy score, the percentages of the correctly classified training, validation, and test images are computed.

4.2. Additive manufacturing

The AM dataset is taken from the RAISE-LPBF benchmarking dataset [4], which includes a selection of high-speed video recordings at 20,000 frames per second of a laser powder bed fusion processes for stainless steel. The laser power and speed parameters are systematically varied. The goal is to reconstruct the power and speed of the laser from this video input. By comparing the predicted laser parameters with the pre-set parameters of the machine producing the laser, anomalies in the printing process can be detected faster, leading to more efficient quality control. The base ML model used for this task is a Swin-Transformer [44], with the HPO search space consisting of multiple, Transformer-specific architectural and optimizer-related parameters, such as the number of attention heads, see Table 1. The model is trained on the C027 cylinder with a 80/20 split for training and validation and is approximately 60 GB in size. It is evaluated on the C028 cylinder for testing purposes. The Mean-Squared Error (MSE) between predicted and actual laser power and speed is computed to assess the accuracy of the SwinTransformer. All models are trained for min_t = 5 to max_t = 20 epochs and a reduction and scaling factor `sf` = `rf` = 2 is chosen for the schedulers. Following Eq. (3), this results in rung milestones at epochs 5, 10, and 20.

4.3. Computational fluid dynamics

The CFD dataset contains actuated turbulent boundary layer flow data, generated from a simulation [5]. The CFD dataset is stored in HDF5 file format and comprises several widths. In this study, widths of 1000, 1200, and 1600 are used as training dataset (approximately 4.8 TB in size). Width of 1800 and 3000 are used as validation and test datasets (approximately 3.5 TB in size) to assess extrapolation performance. Altogether, the dataset is approximately 8.3 TB in size.

A convolutional autoencoder, selected from the AI4HPC repository [45,46], is employed for flow reconstruction. The autoencoder comprises an encoder, a decoder, and a latent space representing a compressed, lower-dimensional version of the input. Both the encoder and decoder include four convolutional layers. In the encoder, the initial two layers perform down-sampling to compress the data, while in the decoder, they perform up-sampling to decompress the data in the latent space. The remaining layers perform regular convolution. The HPO search space consists of the type of activation function as an architectural parameter and several optimizer-related ones, see Table 1.

The autoencoders are trained for min_t = 5 to max_t = 40 epochs and a reduction and scaling factor `sf` = `rf` = 2 is chosen for the schedulers. Following Eq. (3), this results in rung milestones at epochs 5, 10, 20, and 40.

The MSE between the input and the reconstructed output flow field is computed and used to assess the accuracy of the autoencoders. As a further measure of solution quality, also the relative reconstruction error is computed on the test set.

All experiments use reduction and scaling factors `sf` = `rf` = 2, as this provides a suitable trade-off between terminating unpromising trials and scaling up GPU resources. Choosing a scaling factor that is too large can lead to learning instabilities at the rung milestones, due to the abrupt increase in the number of GPUs allocated. Similarly, selecting a reduction factor that is too large may result in prematurely terminating

trials that could have shown strong performance in later training stages. The value $\min\_t = 5$ is selected to ensure sufficient initial training and to avoid interference with the learning rate warm-up phase, which can last up to five epochs (see Table 1). Terminating trials during this period could result in inaccurate early stopping decisions. The values $\max\_t = 20$ and $\max\_t = 40$ are chosen to provide adequate training time for convergence, while also considering computational resource constraints.

5. Results

This section presents the experimental results of running the proposed algorithm on two supercomputer systems, which are introduced in Section 5.1. Section 5.2 focuses on the scaling performance of the RASDA algorithm on up to 1024 GPUs, while Section 5.3 compares the RASDA against the plain ASHA scheduler without any resource adaptation. Section 5.4 reports the performance of RASDA at the large scale, and in Section 5.5 different ablation experiments are presented.

5.1. Supercomputers

The two supercomputer modules used for the experiments in this study are both located at the Jülich Supercomputing Centre.

The first system is the JURECA-DC-GPU module [47] consisting of a total of 192 accelerated compute nodes. Each node is equipped with two AMD EPYC 7742 CPUs with 128 cores clocked at 2.25 GHz and four NVIDIA A100 GPUs, each with 40 GB high-bandwidth memory. The second HPC system is the JUWELS BOOSTER module [48] consisting of a total of 936 compute nodes. Each node is equipped with two AMD EPYC Rome 7402 CPUs with 48 cores clocked at 2.8 GHz, and four NVIDIA A100 GPU with 40 GB high-bandwidth memory. The main difference between the two systems is the number of InfiniBand interconnects: the JURECA-DC-GPU system features only two per node, while the JUWELS BOOSTER has four per node and therefore a higher network transmission bandwidth.

As of June 2024, both supercomputers are among the top 10% most energy-efficient supercomputers in the world, according to the GREEN500 list.[5]

5.2. Scaling performance

To evaluate the scalability of the RASDA algorithm, two weak scaling experiments, where the number of HPO configurations to evaluate is increased with the number of GPUs, are conducted with a lower number of training epochs. For this purpose, the CV application case as a representative benchmark for DL workloads is selected. It should be noted that while the asynchronous nature of the plain ASHA algorithm naturally leads to good scalability [7], the goal of this study is to demonstrate that the additional resource allocation mechanism in RASDA maintains this favorable scaling behavior.

The first weak scaling experiment considers a smaller scale of 8 to 64 GPUs. The runtime and accuracy of the RASDA algorithm is compared to the plain ASHA algorithm for training a ResNet50 on the ImageNet dataset for 20 epochs, see Fig. 3. It can be seen that on all scales (from 8 to 64 GPUs), the RASDA algorithm achieves consistently lower runtimes up to a factor of 1.45 faster than its ASHA counterpart while matching the final test set accuracy in almost all cases. Reduced test set accuracy is observed only at 8 and, to a lesser extent, 16 GPUs, as in these small settings the number of hyperparameter configurations evaluated is low and RASDA cannot yet fully benefit from large batch size training (see Section 3.2).

The second scaling experiment considers a large scale of 128 to 1024 GPUs, see Fig. 4. The weak scalability of the RASDA algorithm is evaluated by training a ResNet for six epochs. The results show that the algorithm maintains a high parallel efficiency of >0.84 on up to 1024 GPUs.

It should be noted that strong scaling experiments that keep the number of hyperparameter configurations consistent across all scales are generally infeasible for this type of HPO workload, as evaluating a large number of configurations on a small number of GPUs would take too long.

The better scaling performance of RASDA compared to ASHA can also be observed when examining specific HPC metrics. RASDA consistently achieves higher GPU utilization, as it avoids idling GPUs, see Fig. 2. RASDA achieves approximately ≈80% GPU utilization, compared to around ≈54% for the ASHA case. Since more GPUs are actively used, this also leads to higher I/O demand compared to ASHA, where idle GPUs consume less data. However, because RASDA achieves significantly shorter runtimes, it is expected to be more energy-efficient overall.

5.3. Speed-ups and accuracy

To evaluate the performance of the RASDA algorithm in terms of speed-up and accuracy and to juxtapose it to the plain ASHA algorithm considering the application cases, the number of training epochs is increased within the $\min\_t$ and $\max\_t$ range specified in Section 4. The general results for the three application case, averaged over three different runs for all application cases, are presented in Tables 2, 3, and 4. The solution quality over time is presented in Fig. 5, an in-depth performance analysis of the runtimes per epoch is given in Fig. 6, and the change of batch size and number of GPUs per trial is depicted in Fig. 7. The results correspond to an exemplary best-performing trial from one of the three runs. The following paragraphs provide a more detailed discussion of these tables and figures.

For the CV application case, a total of 32 hyperparameter combinations are evaluated simultaneously on 64 GPUs on the JURECA-DC-GPU system, with each parallel trial starting with two GPUs. Compared to the plain ASHA approach, the RASDA algorithm reduces the overall average runtime of the HPO process by a factor of ≈1.71 from 527 to 308 min, see Table 2. The average solution quality, i.e., the training, validation, and test set accuracy of the best trial discovered during the process, slightly outperforms the ones of the plain ASHA. This indicates that scaling the batch size and the learning rate during the training process does not impact the learning process in this case. A closer look at one of the best-performing trials in Fig. 6 reveals that indeed the average runtime decreases in the RASDA case once the resource adaptation in space sets in after the first five epochs. As can be seen in Fig. 7, BS_{global} increases from 256 to 2048 during the training and the number of GPUs from 2 to 16 per trial for the RASDA case, while both stay constant in the plain ASHA case. The plot of the validation accuracy over the number of epochs in Fig. 5 confirms that RASDA slightly outperforms the ASHA approach in terms of solution quality.

For the AM application case, the HPO process evaluates 16 configurations, using a total of 128 GPUs on the JURECA-DC-GPU system. The trials start out with 8 GPUs each, which increases to 32 GPUs for the top-performing trials, at the same time increasing BS_{global} from 64 to 256. As the models are only trained for a total amount of 20 epochs (due to the long training times of transformer models), only two resource-doubling steps, i.e., at epoch 5 and epoch 10, take place, see Fig. 7. Table 3 provides an overview of the results in terms of runtime and solution quality. In comparison with the plain ASHA algorithm, a speed-up by a factor of 1.52 is achieved, reducing the required HPO runtime of the models from 96 to 63 min. On both the validation and test dataset, the best configuration found by RASDA again outperforms the one found with the plain ASHA after 20 epochs, as can be seen in Fig. 5.

[5] GREEN500: https://top500.org/lists/green500/list/2024/06/.

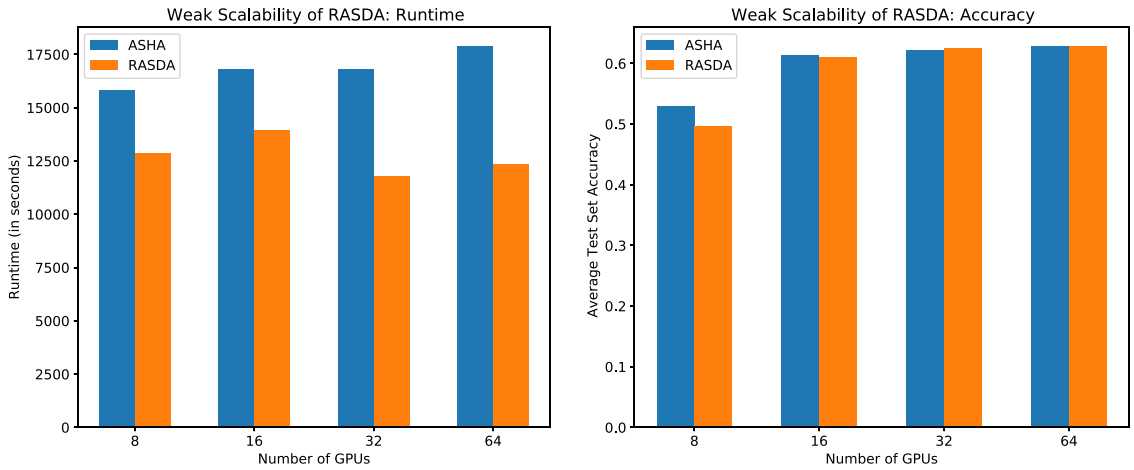


Fig. 3. Comparison of ASHA and RASDA for training a ResNet50 model on ImageNet for 20 epochs on different scales on the JURECA-DC-GPU system.

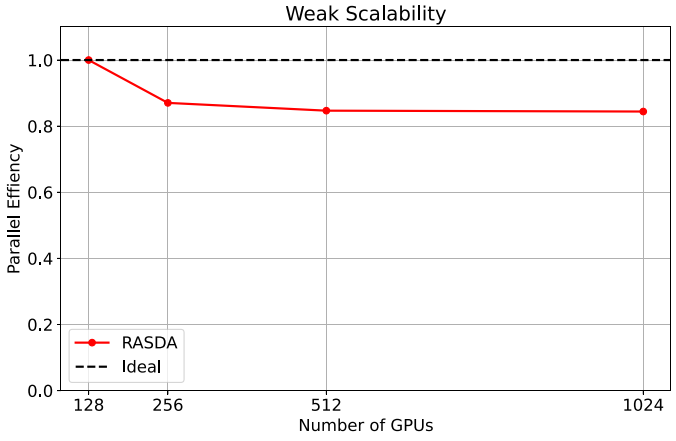


Fig. 4. Weak scalability of the RASDA algorithm on up to 1024 GPUs on the JUWELS BOOSTER system, including ideal scalability for comparison.

The CFD application case features the largest dataset used in this study. The whole HPO process evaluates 16 configurations on 128 GPUs simultaneously on the JUWELS BOOSTER module. Each trial starts with 8 GPUs, which is increased over time to 64 GPU by the RASDA algorithm. As can be seen from Table 4, the most significant speed-up with a factor of ≈1.9 is achieved in this case, with RASDA reducing the runtime of the HPO process from 325 to 170 min. In this case, also the average MSE decreases by a factor of ≈1.88. This is likely due to the even better generalization capabilities caused by increasing the batch size over time (following the insights explained in Section 3.2). Obviously, this outperforms just annealing of the learning rate. This observation is in line with the findings of Smith et al. [33]. RASDA also achieves a low relative reconstruction error of just 1.15% on the test set.

In general, the most substantial speed-up is established on the largest dataset from the CFD domain. This is expected, as with a larger dataset, the benefit of adding more GPUs to the data-parallel training loop also increases. It is additionally interesting to observe that the speed-ups can be attained on both the JURECA-DC-GPU and JUWELS BOOSTER systems, although the latter features twice the network bandwidth. While RASDA already yields substantial benefits on JURECA-DC-GPU with its moderate network infrastructure, the doubled network bandwidth of JUWELS BOOSTER further amplifies these speed-ups, highlighting how the approach particularly profits from fast interconnects.

Table 2
HPO for the CV application case, trained for 40 epochs on 64 GPUs on the JURECA-DC-GPU system. Results are averaged over three random seeds. Better results (↑ or ↓ depending on the metric) are underlined.

Metric	ASHA	RASDA	Diff.
Train accuracy ↑	0.6976	0.7310	1.05×
Val accuracy ↑	0.6728	0.6813	1.01×
Test accuracy ↑	0.6688	0.6766	1.01×
Runtime (in seconds) ↓	31 637	18 502	1.71×

Table 3
HPO for the AM application case, trained for 20 epochs on 128 GPUs on the JURECA-DC-GPU system. Results are averaged over three random seeds. Better results (↑ or ↓ depending on the metric) are underlined. For better comparison the metrics were recomputed on a per-sample basis after the run.

Metric	ASHA	RASDA	Diff.
Val MSE ↓	0.0455	0.0404	1.12×
Test MSE ↓	0.0554	0.0516	1.07×
Runtime (in seconds) ↓	5784	3803	1.52×

Table 4
HPO for the CFD application case, trained for 40 epochs on 128 GPUs on the JUWELS BOOSTER module. Results are averaged over three random seeds. Better results (↑ or ↓ depending on the metric) are underlined.

Metric	ASHA	RASDA	Diff.
Val MSE ↓	5.28×10^{-6}	2.81×10^{-6}	1.88×
Test MSE ↓	4.42×10^{-6}	2.40×10^{-6}	1.84×
Test relative error ↓	0.0185	0.0115	1.61×
Runtime (in seconds) ↓	19 487	10 242	1.90×

5.4. Performance at 1024 GPUs scale

While the superiority of RASDA over plain ASHA has been confirmed in the previous experiments using 64 and 128 GPUs, a final RASDA experiment on a 1024 GPU scale is conducted on the JUWELS BOOSTER system. Again, using the CFD application case, the number of configurations to be evaluated is increased to 64, with each trial starting with 16 GPUs. The models are trained for $\min\_t = 5$ and $\max\_t = 20$ epochs. The HPO run took three hours and resulted in an improved model with a validation MSE of $\approx 3.63 \times 10^{-7}$, a test MSE of $\approx 4.88 \times 10^{-8}$ and a relative test error of ≈0.0016. Depending on the metric, this is a 7 to 49 times increase in solution quality, compared to

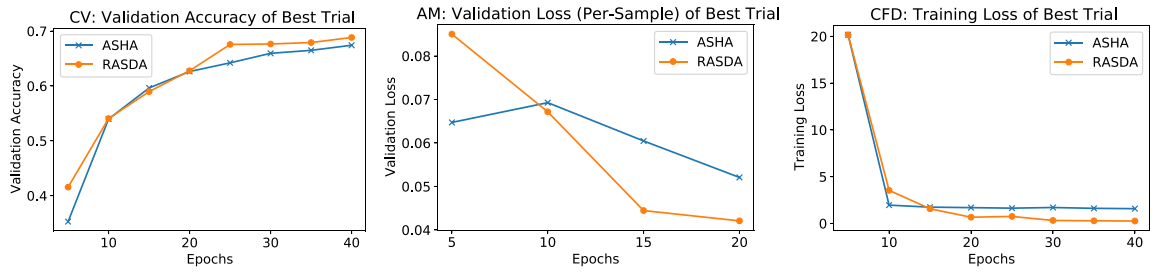


Fig. 5. Exemplary comparison of the performance (in terms of validation accuracy, training loss, and validation loss) of the best configuration found by ASHA and RASDA for the different application cases.

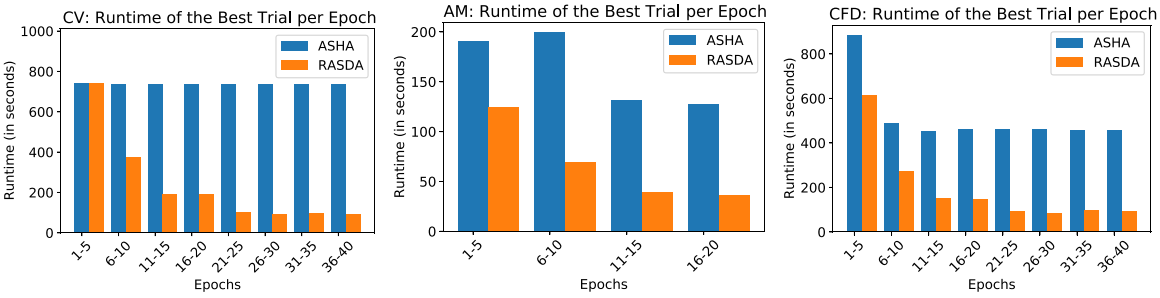


Fig. 6. Exemplary comparison of the runtime per epoch of the best configuration found by ASHA and RASDA for the different application cases. Note that for the CFD and AM case, the architectural parameters chosen by the respective HPO method also influence the model size, which is why here differences in runtime can be observed already during the first five epochs.

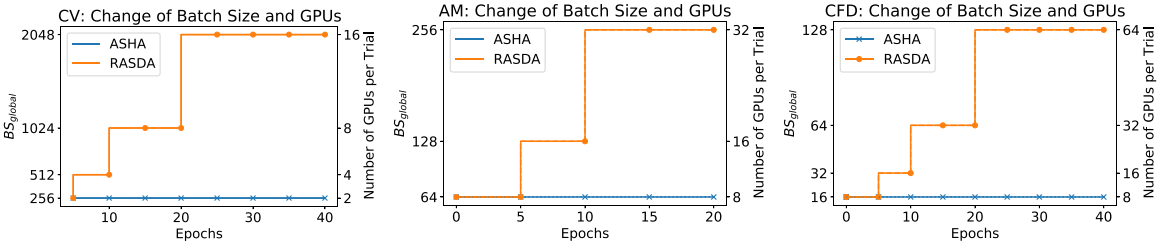


Fig. 7. Comparison of the global batch size and the number of GPUs per trial for ASHA and RASDA for the different application cases.

Table 5
Large-scale HPO for the CFD application case, evaluating 64 configurations, trained for a maximum of 20 epochs on 1024 GPUs on the JUWELS BOOSTER module, including relative improvement to the HPO run on 128 GPUs.

Metric	RASDA - 1024 GPUs	vs. 128 GPUs
Val MSE	3.63×10^{-7}	7.74×
Test MSE	4.88×10^{-8}	49.22×
Test Rel. error	0.0016	7.17×

Table 6
Comparison of ASHA and RASDA with `sf` = `rf` = 4 for the CV application case, trained for 20 epochs on 64 GPUs on the JURECA-DC-GPU system. Results are averaged over three random seeds. Better results (↑ or ↓ depending on the metric) are underlined.

Metric	ASHA	RASDA	Diff.
Train accuracy ↑	0.6179	<u>0.6556</u>	1.06×
Val accuracy ↑	0.6312	<u>0.6348</u>	1.01×
Test accuracy ↑	0.6250	<u>0.6340</u>	1.01×
Runtime (in seconds) ↓	17 830	<u>11 157</u>	1.60×

the results of the HPO run on 128 GPUs (see Table 5), which highlights the potential of large-scale HPO for scientific ML.

5.5. Ablation studies

To evaluate the impact of different parameters on the performance of the RASDA method, two ablation studies are conducted.

5.5.1. Impact of reduction and scaling factors

All prior experiments use a scaling and reduction factor of `sf` = `rf` = 2. In this ablation study on the CV application case, these factors are increased to `sf` = `rf` = 4. Since the number of GPU accelerators per node on HPC systems typically follows a power-of-two configuration, allocating partial nodes may lead to performance degradation. To ensure sufficient training time between decision points, the same values of $\min\_t = 5$ and $\max\_t = 20$ epochs are retained. According to Eq. (3), this results in two rung milestones at epochs 5 and 20. As shown in Table 6, RASDA continues to outperform ASHA under these settings, achieving higher accuracy and a runtime speed-up of ≈1.6×. However, this speed-up is smaller than the improvement observed with `sf` = `rf` = 2 (see Table 2). While a larger scaling factor

Table 7
Comparison of RASDA and BOHB on the CV application case, trained for 20 epochs on 64 GPUs on the JURECA-DC-GPU system, evaluating 32 hyperparameter samples. Results are averaged over three random seeds. Better results (↑ or ↓ depending on the metric) are underlined.

Metric	BOHB	RASDA	Diff.
Train accuracy ↑	0.6130	<u>0.6480</u>	1.06×
Val accuracy ↑	0.6253	<u>0.6271</u>	1.00×
Test accuracy ↑	0.6222	<u>0.6254</u>	1.01×
Runtime (in seconds) ↓	16 825	<u>11 815</u>	1.42×

enables more aggressive allocation of GPU resources to promising trials, it also increases the time between rung milestones due to the geometric progression (see Eq. (3)). This observation indicates that `rf = 2` is a suitable choice for the RASDA scheduler.

5.5.2. Comparison to Bayesian Optimization (BO)

Although the primary comparison is between RASDA and its closest scheduling-based counterpart, ASHA, other types of HPO and Neural Architecture Search (NAS) tools are also relevant. One commonly used method is Bayesian Optimization (BO). A comparison between RASDA and the BOHB algorithm on the CV application case is provided below. To function effectively, BO requires the ability to generate new hyperparameter configurations based on past evaluations. As shown in Table 7, when both methods are evaluated with the same number of hyperparameter samples, RASDA outperforms BOHB in terms of accuracy and runtime. These results suggest that RASDA achieves a more favorable trade-off between runtime and solution quality compared to traditional BO-based approaches.

6. Summary and outlook

RASDA, a novel resource-adaptive successive doubling algorithm for HPO, suitable for running on HPC systems, was introduced. The key idea is to not only perform successive halving in time and let promising configurations train for longer (as is already the case in plain ASHA), but to combine it with successive doubling in space and allocate more computational resources to the data-parallel training of promising configurations.

The RASDA method was evaluated extensively on a standard benchmarking task in the CV domain as well as on two large datasets (up to 8.3 TB in size) from the CFD and AM domains. The results confirm that RASDA leads in these cases to speed-ups up to a factor of ≈1.9 in comparison to the ASHA algorithm.

Another property of RASDA is that it progressively scales up the global batch size of the trials as it adds more GPUs to their training loops. This helps them to avoid the degradation in solution quality, which is usually associated with large batch training. Remarkably, the approach did enhance the solution quality, aligning with literature findings suggesting that increasing the batch size can match or surpass the effects of learning rate annealing.

In addition, this study represents the first application of systematic HPO to a scientific dataset at the TB scale. A comparison of the application of RASDA on 128 and 1024 GPUs revealed a significant improvement in model performance. Specifically, the larger-scale application identifies a model that is significantly superior in solution quality. These results demonstrate the scalability and efficiency of the RASDA method, thus paving the way for the application of HPO methods on current and future Exascale supercomputers. It should be noted, however, that RASDA relies heavily on efficient distributed training. As such, its full benefits are realized primarily on HPC systems equipped with accelerators such as GPUs. On non-accelerated systems or in settings with very limited communication bandwidth, potential inefficiencies may limit performance.

For future work, the optimal timing for scaling the batch size (through the addition of more GPUs to the data-parallel training loop) should be investigated more thoroughly. Furthermore, the impact of scaling the batch size on various hyperparameters beyond the learning rate (such as the weight decay values) warrants a deeper exploration.

CRediT authorship contribution statement

Marcel Aach: Writing – original draft, Visualization, Validation, Software, Methodology, Investigation, Conceptualization. **Rakesh Sarma:** Writing – review & editing, Validation, Software, Methodology. **Helmut Neukirchen:** Writing – review & editing, Validation, Supervision. **Morris Riedel:** Writing – review & editing, Validation, Supervision. **Andreas Lintermann:** Writing – review & editing, Validation, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors thank Kurt De Grave and Cyril Blanc for contributing an implementation of the AM use case.

This research has been performed in the CoE RAISE project, which received funding from the European Union's *Horizon 2020 – Research and Innovation Framework Programme* H2020-INFRAEDI-2019-1 under grant agreement no. 951733.

The authors gratefully acknowledge computing time on the supercomputer JURECA [47] at Forschungszentrum Jülich, Germany under grant no. raise-ctp2. The authors gratefully acknowledge the Gauss Centre for Supercomputing e.V. (www.gauss-centre.eu) for funding this project by providing computing time through the John von Neumann Institute for Computing (NIC), Germany on the GCS Supercomputer JUWELS [49] at Jülich Supercomputing Centre (JSC).

Appendix. RASDA GitHub repository

See Table A.8.

Table A.8
Content of the RASDA GitHub Repository.

File	Description
`startscript.sh`	Shell script to launch the HPO process via Ray on an HPC system
`build_ray_env.sh`	Shell script to set up the Ray environment on an HPC system
`adaptive_ray.py`	Main script to configure the schedulers and launch the HPO trials
`res_allocate.py`	Core resource allocation logic for RASDA
`cases/ImagenetTrainLoopDALI.py`	Script containing the training logic of the CV application example

Data availability

All used data is linked in the manuscript.

References

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A.C. Berg, L. Fei-Fei, ImageNet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (3) (2015) 211–252, http://dx.doi.org/10.1007/s11263-015-0816-y.

[2] G. Sumbul, M. Charfuelan, B. Demir, V. Markl, Bigearthnet: A large-scale benchmark archive for remote sensing image understanding, in: IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium, IEEE, 2019, pp. 5901–5904, http://dx.doi.org/10.1109/IGARSS.2019.8900532.

[3] J. Pata, E. Wulff, F. Mokhtar, D. Southwick, M. Zhang, M. Girone, J. Duarte, Improved particle-flow event reconstruction with scalable neural networks for current and future particle detectors, Commun. Phys. 7 (1) (2024/04/10) 124, http://dx.doi.org/10.1038/s42005-024-01599-5.

[4] C. Blanc, A. Ahar, K. De Grave, Reference dataset and benchmark for reconstructing laser parameters from on-axis video in powder bed fusion of bulk stainless steel, Addit. Manuf. Lett. 7 (2023) 100161, http://dx.doi.org/10.1016/j.addlet.2023.100161.

[5] M. Albers, P.S. Meysonnat, D. Fernex, R. Semaan, B.R. Noack, W. Schröder, A. Lintermann, Actuated turbulent boundary layer flows dataset, 2023, http://dx.doi.org/10.34730/5dbc8e35f21241d0889906136cf28d26.

[6] M. Feurer, F. Hutter, Hyperparameter optimization, in: F. Hutter, L. Kotthoff, J. Vanschoren (Eds.), Automated Machine Learning: Methods, Systems, Challenges, Springer International Publishing, Cham, 2019, pp. 3–33, http://dx.doi.org/10.1007/978-3-030-05318-5_1.

[7] L. Li, K. Jamieson, A. Rostamizadeh, E. Gonina, J. Ben-tzur, M. Hardt, B. Recht, A. Talwalkar, A system for massively parallel hyperparameter tuning, in: I. Dhillon, D. Papailiopoulos, V. Sze (Eds.), Proceedings of Machine Learning and Systems 2, MLSys 2020, Vol. 2, 2020, pp. 230–246, URL: https://proceedings.mlsys.org/paper_files/paper/2020/hash/a06f20b349c6cf09a6b171c71b88bbfc-Abstract.html.

[8] J. Bergstra, Y. Bengio, Random search for hyper-parameter optimization, J. Mach. Learn. Res. 13 (2012) 281–305, URL: http://jmlr.org/papers/v13/bergstra12a.html.

[9] K. Jamieson, A. Talwalkar, Non-stochastic best arm identification and hyperparameter optimization, in: A. Gretton, C.C. Robert (Eds.), Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, in: Proceedings of Machine Learning Research, vol. 51, PMLR, 2016, pp. 240–248, URL: https://proceedings.mlr.press/v51/jamieson16.html.

[10] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, A. Talwalkar, Hyperband: A novel bandit-based approach to hyperparameter optimization, J. Mach. Learn. Res. 18 (185) (2018) 1–52, URL: https://jmlr.org/papers/v18/16-558.html.

[11] S. Falkner, A. Klein, F. Hutter, BOHB: Robust and efficient hyperparameter optimization at scale, in: Proceedings of the 35th International Conference on Machine Learning, Vol. 80, PMLR, 2018, pp. 1436–1445, URL: https://proceedings.mlr.press/v80/falkner18a.html.

[12] J. Bergstra, R. Bardenet, Y. Bengio, B. Kégl, Algorithms for hyper-parameter optimization, in: J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, K.Q. Weinberger (Eds.), Proceedings of the 24th International Conference on Neural Information Processing Systems, NIPS '11, Vol. 24, Curran Associates, Inc., 2011.

[13] R. Liaw, R. Bhardwaj, L. Dunlap, Y. Zou, J.E. Gonzalez, I. Stoica, A. Tumanov, HyperSched: Dynamic resource reallocation for model development on a deadline, in: Proceedings of the ACM Symposium on Cloud Computing, SoCC '19, ACM, 2019, pp. 61–73, http://dx.doi.org/10.1145/3357223.3362719.

[14] U. Misra, R. Liaw, L. Dunlap, R. Bhardwaj, K. Kandasamy, J.E. Gonzalez, I. Stoica, A. Tumanov, RubberBand: cloud-based hyperparameter tuning, in: Proceedings of the Sixteenth European Conference on Computer Systems, EuroSys '21, ACM, 2021, pp. 327–342, http://dx.doi.org/10.1145/3447786.3456245.

[15] L. Dunlap, K. Kandasamy, U. Misra, R. Liaw, M. Jordan, I. Stoica, J.E. Gonzalez, Elastic hyperparameter tuning on the cloud, in: Proceedings of the ACM Symposium on Cloud Computing, SoCC '21, ACM, 2021, pp. 33–46, http://dx.doi.org/10.1145/3472883.3486989.

[16] K. Kandasamy, K.R. Vysyaraju, W. Neiswanger, B. Paria, C.R. Collins, J. Schneider, B. Poczos, E.P. Xing, Tuning hyperparameters without grad students: Scalable and robust Bayesian optimisation with dragonfly, J. Mach. Learn. Res. 21 (81) (2020) 1–27, URL: http://jmlr.org/papers/v21/18-223.html.

[17] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, D. Deng, C. Benjamins, T. Ruhkopf, R. Sass, F. Hutter, SMAC3: A versatile Bayesian optimization package for hyperparameter optimization, J. Mach. Learn. Res. 23 (54) (2022) 1–9, URL: http://jmlr.org/papers/v23/21-0888.html.

[18] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: A next-generation hyperparameter optimization framework, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD '19, ACM, 2019, pp. 2623–2631, http://dx.doi.org/10.1145/3292500.3330701.

[19] P. Balaprakash, M. Salim, T.D. Uram, V. Vishwanath, S.M. Wild, DeepHyper: Asynchronous hyperparameter search for deep neural networks, in: 2018 IEEE 25th International Conference on High Performance Computing, IEEE, 2018, pp. 42–51, http://dx.doi.org/10.1109/HiPC.2018.00014.

[20] P. Balaprakash, R. Egele, M. Salim, S. Wild, V. Vishwanath, F. Xia, T. Brettin, 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, ACM, 2019, http://dx.doi.org/10.1145/3295500.3356202.

[21] S. Jiang, P. Balaprakash, Graph neural network architecture search for molecular property prediction, in: 2020 IEEE International Conference on Big Data, Big Data, IEEE, 2020, pp. 1346–1353, http://dx.doi.org/10.1109/BigData50022.2020.9378060.

[22] X. Liu, M. Rüttgers, A. Quercia, R. Egele, E. Pfaehler, R. Shende, M. Aach, W. Schröder, P. Balaprakash, A. Lintermann, Refining computer tomography data with super-resolution networks to increase the accuracy of respiratory flow simulations, Future Gener. Comput. Syst. 159 (2024) 474–488, http://dx.doi.org/10.1016/j.future.2024.05.020.

[23] O. Taubert, M. Weiel, D. Coquelin, A. Farshian, C. Debus, A. Schug, A. Streit, M. Götz, Massively parallel genetic optimization through asynchronous propagation of populations, in: A. Bhatele, J. Hammond, M. Baboulin, C. Kruse (Eds.), High Performance Computing. ISC High Performance 2023, in: LNCS, vol. 13948, Springer, 2023, pp. 106–124, http://dx.doi.org/10.1007/978-3-031-32041-5_6.

[24] M. Jaderberg, V. Dalibard, S. Osindero, W.M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, C. Fernando, K. Kavukcuoglu, Population based training of neural networks, 2017, arXiv:1711.09846.

[25] S. Li, Y. Zhao, R. Varma, O. Salpekar, P. Noordhuis, T. Li, A. Paszke, J. Smith, B. Vaughan, P. Damania, S. Chintala, PyTorch distributed: Experiences on accelerating data parallel training, Proc. Very Large Data Base Endow. 13 (12) (2020) 3005–3018, http://dx.doi.org/10.14778/3415478.3415530.

[26] P. Goyal, P. Dollár, R. Girshick, P. Noordhuis, L. Wesolowski, A. Kyrola, A. Tulloch, Y. Jia, K. He, Accurate, large minibatch SGD: Training ImageNet in 1 hour, 2017, http://dx.doi.org/10.48550/ARXIV.1706.02677, arXiv:1706.02677.

[27] S. Hochreiter, J. Schmidhuber, Flat minima, Neural Comput. 9 (1) (1997) 1–42, http://dx.doi.org/10.1162/neco.1997.9.1.1.

[28] N.S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, P.T.P. Tang, On large-batch training for deep learning: Generalization gap and sharp minima, 2017, http://dx.doi.org/10.48550/arXiv.1609.04836, arXiv:1609.04836.

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

[30] S. McCandlish, J. Kaplan, D. Amodei, O.D. Team, An empirical model of large-batch training, 2018, arXiv:1812.06162.

[31] A. Qiao, S.K. Choe, S.J. Subramanya, W. Neiswanger, Q. Ho, H. Zhang, G.R. Ganger, E.P. Xing, Pollux: Co-adaptive cluster scheduling for goodput-optimized deep learning, in: 15th USENIX Symposium on Operating Systems Design and Implementation, OSDI 21, USENIX Association, 2021, pp. 1–18, URL: https://www.usenix.org/conference/osdi21/presentation/qiao.

[32] M. Aach, R. Sedona, A. Lintermann, G. Cavallaro, H. Neukirchen, M. Riedel, Accelerating hyperparameter tuning of a deep learning model for remote sensing image classification, in: IGARSS 2022 - 2022 IEEE International Geoscience and Remote Sensing Symposium, IEEE, 2022, pp. 263–266, http://dx.doi.org/10.1109/IGARSS46834.2022.9883257.

[33] S.L. Smith, P.-J. Kindermans, Q.V. Le, Don't decay the learning rate, increase the batch size, in: International Conference on Learning Representations, 2018, URL: https://openreview.net/pdf?id=B1Yy1BxCZ.

[34] I. Loshchilov, F. Hutter, SGDR: Stochastic gradient descent with warm restarts, in: International Conference on Learning Representations, 2017, URL: https://openreview.net/pdf?id=Skq89Scxx.

[35] A. Krizhevsky, One weird trick for parallelizing convolutional neural networks, 2014, arXiv:1404.5997.

[36] S. Malladi, K. Lyu, A. Panigrahi, S. Arora, On the SDEs and scaling rules for adaptive gradient algorithms, in: S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, A. Oh (Eds.), Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS '22, Curran Associates, Inc., 2022, URL: https://proceedings.neurips.cc/paper_files/paper/2022/file/32ac710102f0620d0f28d5d05a44fe08-Paper-Conference.pdf.

[37] M. Aach, R. Sarma, E. Inanc, M. Riedel, A. Lintermann, Short paper: Accelerating hyperparameter optimization algorithms with mixed precision, in: Proceedings of the SC '23 Workshops of the International Conference on High Performance Computing, Network, Storage, and Analysis, in: SC-W '23, ACM, 2023, pp. 1776–1779, http://dx.doi.org/10.1145/3624062.3624259.

[38] R. Martinez-Cantin, BayesOpt: A Bayesian optimization library for nonlinear optimization, experimental design and bandits, J. Mach. Learn. Res. 15 (115) (2014) 3915–3919, URL: http://jmlr.org/papers/v15/martinezcantin14a.html.

[39] A.I. Cowen-Rivers, W. Lyu, R. Tutunov, Z. Wang, A. Grosnit, R.R. Griffiths, A.M. Maraval, H. Jianye, J. Wang, J. Peters, H. Bou-Ammar, HEBO: Pushing the limits of sample-efficient hyper-parameter optimisation, J. Artif. Int. Res. 74 (2022) http://dx.doi.org/10.1613/jair.1.13643.

[40] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification, in: Proceedings of the 2015 IEEE International Conference on Computer Vision, ICCV, IEEE, 2015, pp. 1026–1034, http://dx.doi.org/10.1109/ICCV.2015.123.

[41] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in: Y.W. Teh, M. Titterington (Eds.), Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, in: Proceedings of Machine Learning Research, vol. 9, PMLR, 2010, pp. 249–256, URL: https://proceedings.mlr.press/v9/glorot10a.html.

[42] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR, 2016, pp. 770–778, http://dx.doi.org/10.1109/CVPR.2016.90.

[43] P. Mattson, C. Cheng, G. Diamos, C. Coleman, P. Micikevicius, D. Patterson, H. Tang, G.-Y. Wei, P. Bailis, V. Bittorf, D. Brooks, D. Chen, D. Dutta, U. Gupta, K. Hazelwood, A. Hock, X. Huang, D. Kang, D. Kanter, N. Kumar, J. Liao, D. Narayanan, T. Oguntebi, G. Pekhimenko, L. Pentecost, V. Janapa Reddi, T. Robie, T. St John, C.-J. Wu, L. Xu, C. Young, M. Zaharia, MLPerf training benchmark, in: I. Dhillon, D. Papailiopoulos, V. Sze (Eds.), Proceedings of Machine Learning and Systems, Vol. 2, 2020, pp. 336–349, URL: https://proceedings.mlsys.org/paper_files/paper/2020/file/411e39b117e885341f25efb8912945f7-Paper.pdf.

[44] Y.-Q. Yang, Y.-X. Guo, J.-Y. Xiong, Y. Liu, H. Pan, P.-S. Wang, X. Tong, B. Guo, Swin3D: A pretrained transformer backbone for 3D indoor scene understanding, 2023, arXiv:2304.06906.

[45] E. Inanc, R. Sarma, M. Aach, A. Lintermann, AI4HPC, 2023, http://dx.doi.org/10.5281/zenodo.7705417.

[46] E. Inanc, R. Sarma, M. Aach, R. Sedona, A. Lintermann, AI4HPC: Library to train AI models on HPC systems using CFD datasets, in: Workshop on Advancing Neural Network Training: Computational Efficiency, Scalability, and Resource Optimization, WANT@NeurIPS 2023, 2023, URL: https://openreview.net/pdf?id=zQTa2XdPnP.

[47] Jülich Supercomputing Centre, JURECA: Data centric and booster modules implementing the modular supercomputing architecture at Jülich Supercomputing Centre, J. Large- Scale Res. Facil. JLSRF 7 (2021) http://dx.doi.org/10.17815/jlsrf-7-182.

[48] S. Kesselheim, A. Herten, K. Krajsek, J. Ebert, J. Jitsev, M. Cherti, M. Langguth, B. Gong, S. Stadtler, A. Mozaffari, G. Cavallaro, R. Sedona, A. Schug, A. Strube, R. Kamath, M.G. Schultz, M. Riedel, T. Lippert, JUWELS booster – a supercomputer for large-scale AI research, in: H. Jagode, H. Anzt, H. Ltaief, P. Luszczek (Eds.), High Performance Computing, in: LNCS, vol. 12761, Springer, 2021, pp. 453–468, http://dx.doi.org/10.1007/978-3-030-90539-2_31.

[49] Jülich Supercomputing Centre, JUWELS Cluster and Booster: Exascale Pathfinder with Modular Supercomputing Architecture at Juelich Supercomputing Centre, J. Large- Scale Res. Facil. JLSRF 7 (2021) http://dx.doi.org/10.17815/jlsrf-7-183.