Universiteit Leiden

# Optimization Algorithm Ranking Preservation Using Surrogate Models in the MECHBench Framework

Odysseas Lydakis Simantiris (s3872718)

Supervisors:
Dr. Elena Raponi & Iván Olarte Rodríguez MSc

Bachelor Thesis– Data Science and Artificial Intelligence

Leiden Institute of Advanced Computer Science (LIACS)
liacs.leidenuniv.nl

July 12, 2026

**Abstract**

Optimization techniques for problems in structural mechanics are an active point of research, due to the limitations of systems in the field which make it difficult to implement optimizers with satisfactory performance. Thus, optimization benchmarking suites are integral for providing a framework on which to test different approaches to optimization of such problems. A technique some suites use to be applicable to this situation is simulation-based solvers, which are computationally expensive. This study investigates the possibility of implementing surrogate-based alternatives based on an existing simulation-based benchmarking suite, to replace the simulation tool with a computationally cheaper solution. This is done by testing several model classes, and for each class, separate instances trained on multiple dataset sizes. The suitability of each tested model class is analyzed through the lens of comparing the ranking of optimization algorithms between the model and simulation based approaches. By analyzing the results of this experiment process, it remains inconclusive whether the surrogate models can preserve algorithm ranking, but it is found that the size of the training set is not relevant to the degree of preservation.

# Contents

# 1 Introduction

Optimization is crucial to the field of structural mechanics, since it is required for designing structures that satisfy requirements that support their purpose. One of the most important application cases is to use optimization to satisfy constraints in weight and ensure crash-resistance of structures for vehicle crashworthines, where optimality is crucial for human user safety[6]. Nonetheless, due to the nature of systems in structural mechanics there is still debate around which computational methods are satisfactory to create methods which facilitate the optimization process in the field, with both gradient-based and gradient-free methods exhibiting significant drawbacks. Therefore, the selection of techniques for such optimization tasks is an open and current point of research.

For this purpose of testing the performance of optimization techniques, benchmarking suites are used to conduct experiments using these algorithms[25]. One such suite is MECHBench[32], for which Rodriguez et al. use OpenRadioss[1] as a simulation tool in order to perform optimization tasks in three defined representative systems from structural mechanics. The inner workings of the suite, as well as explanations on the systems it defines and works with, will be further discussed in Section 2.4.

In this study, as part of a larger project to test computationally cheaper alternatives to the expensive OpenRadioss simulation used in MECHBench, different classes of surrogate models will be implemented and integrated into the codebase of the suite. Model instances for each class will be trained on different sized datasets, and these will be tested on their preservation of optimization algorithm ranking between the models and the simulation based versions of the suite.

## 1.1 Research Questions

This research project studies a specific application case, but does so within the broader endeavor of researching the feasibility of computationally cheaper alternatives to EFEM simulations for optimization benchmarks for real-world problems, particularly in the field of structural mechanics.

With this context, we define the Research Question **(RQ)** which we will attempt to answer, as well as a secondary sub-RQ as follows:

**RQ:** Is optimization algorithm ranking preserved between the simulation-based and a proposed surrogate model-based version of the MECHBench optimization benchmark suite?

**sub-RQ:** How does the ranking preservation vary when the used surrogate-models are trained on different-sized datasets ($30/60/125/250/500 \times n_{\text{dimensions}}$)?

## 1.2 Hypotheses

Based on our RQs we define two hypotheses:

- $\mathbf{H_1}$**:** Algorithm ranking is not preserved between the modeled objective function and the MECHBench objective function.

- **$H_2$:** There exists a linear relation between the training set size of the models, and the degree to which algorithm ranking is preserved.

### 1.3 Thesis Overview

This thesis will go over the background and related work of the involved concepts including optimization benchmarking suites, the advantages and concerns, as well as the findings of works on using surrogate models in optimization benchmarks, the limitations of optimization in structural mechanics and an analysis of the inner workings of the MECHBench suite (Section 2). In Section 3 the specific configuration of the used MECHBench problems is provided, along with details on how the surrogate models were implemented and why these classes were chosen, as well as an explanation of the used optimizer algorithms and how they are evaluated, followed by an outline of our experiment design. This Section ends with a description of the entire flow of this project. Following are the Results and Discussion sections (Sections 4 & 5), where the results produced by the experiment are reported with plots, and are interpreted and compared to literature in order to analyze the gained insight. Section 6 goes over the limitations of this study, and potential suggestions for future work on this topic. Finally, in Section 7 our observations as discussed in the previous sections, are put together to formulate our final conclusions regarding our RQs and hypotheses.

## 2 Background & Related Work

### 2.1 Optimization Benchmarking Suites

Generally, black-box optimizers, are tested by performing experiments using synthetic optimization benchmarking suites[25], such as COCO[17], which are often implemented with focus on computational efficiency. However, when it comes to systems in real-world optimization tasks, such as problems in structural mechanics, which exhibit high complexity, when using mathematical representations of the systems it becomes uncertain whether the important information contained in this complexity will be captured, potentially leading to unreliable results by the optimization benchmark[5]. Benchmarking suites created to address this issue often utilize simulation tools[39], sacrificing computational efficiency for accuracy. That being said, there is a gap in such software for optimization processes on structural mechanics systems, with existing benchmarks for problems related to the field utilizing different techniques[21][24]. The MECHBench suite provides a framework for problems in structural mechanics utilizing the OpenRadioss Explicit Finite Element Method (EFEM) simulation tool[32][1].

### 2.2 Surrogate Model-Based Approaches

The approach of using surrogate models for such optimization benchmarks has the advantage that the models are trained on datasets from the real world problem, but in contrast to using simulation tools, once the model is trained it is computationally cheap to perform many evaluations[5]. One such implementation, presented by Don Jones at the MOPTA 2008 conference[21], involves training model instances on high fidelity training sets obtained from first running Finite Element (FE) simulations on the problem of interest. This means that overall, the bulk of computational expense

can be on creating a dataset on which to fit the models, which can be saved and used to perform many evaluations quickly at minimal computational cost[20].

In their work, Bliek et al. [5], test surrogate models with optimization algorithms for real-world expensive optimization tasks. They report that in their tested cases, exploration of the model is important for the performance, so offline learning, that is, training the models on datasets of the objective function separately from the optimization process was found to outperform the alternative. This approach is also the one taken to training the models for this project.

Furthermore, Fang et al.[14] find that surrogate modeling techniques are the most feasible, and provide a fitting approach to such problems in lower dimensions, but suffer from the curse of dimensionality, as performance does not remain satisfactory when scaling the amount of variables.

## 2.3 Limitations in Structural Mechanics

As is mentioned in sections above, the problems in structural mechanics, make it difficult to create a computational approach to perform optimization tasks reliably and conveniently. In specific, gradient-based methods are cumbersome to implement correctly and often underperforming due to the highly non-linear nature of these problems. The complexity of the mathematical representations creates energy landscapes that require particularly small stepsizes in gradient calculation[14] and information of the landscape is lost due to noise when using finite differences operations to compute the gradient[38]. Moreover, gradient-free methods, while shown to provide accurate results when combined with EFEMs[11], are still debated for their suitability for this application[32], and come with the drawback of needing a lot of computational resources to be implemented, since most gradient-free optimizers require multiple evaluations of a problem[26], which is computationally expensive when using EFEM simulators[27].

## 2.4 MECHBench

MECHBench[32] is a black-box optimization benchmarking suite for the field of structural mechanics that is the core focus of this thesis project. The authors of the paper proposing the suite choose three problems, representative of systems in structural mechanics, on which to apply their proposed process, two of which are used in this project. Each optimization problem has a defined meaning for each dimension in its configuration, as well as a mathematical formulation based on an objective function depending on certain objective values. A detailed description of the two MECHBench problems relevant to this study, including their specific mathematical formulations, is provided below, in Section 2.4.1.
An optimization loop using MECHBench, as proposed in the related paper[32], and shown in Figure 1 is described as follows: Start with a randomized vector of values in the range $[-5, 5]$ which represent the value of each defined variable of the problem of interest. The values in the vector are used to assemble the according model, that is, the structure in the configuration given by these values. This model is passed to the OpenRadioss [1] engine which simulates the corresponding scenario (e.g. compression by an impactor). This process is logged, and returns extracted objective values which are chosen by the user. Finally the objective values are passed to an optimization

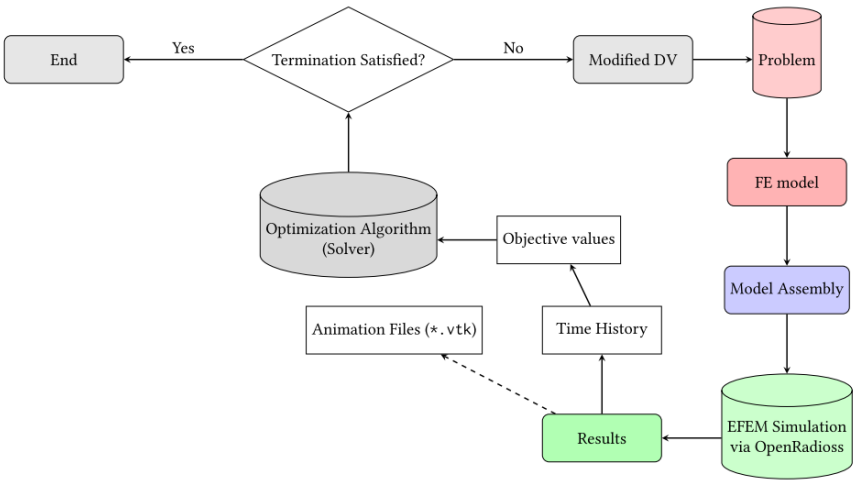



Figure 1: Optimization loop using MECHBench (from [32])

algorithm, if the termination constraint for the loop is satisfied, the current configuration is kept and otherwise the vector values as they are at this point get passed to a new loop iteration.
The suite manages to provide a modular and stable framework for applying optimization processes to structural mechanics. However, due to the use of the OpenRadioss and the related EFEM processes, a single evaluation of each of the problems requires at least a minute of compute, with most configurations requiring significantly more, even with the use of multi-threaded processes[32]. This means that the amount of compute time needed for running experiments or optimization processes which normally require several evaluations becomes very large. In this thesis work, as described in Section 3, we attempt to replace the FE processes of constructing a model and evaluating it in a simulation each time, with surrogate models.

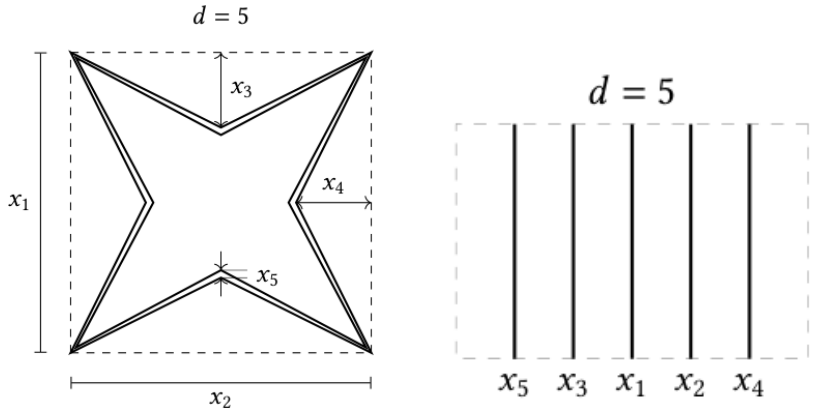
### 2.4.1 Problems in Structural Mechanics

**MECHBench Problem 1 (MB1): Star-Shaped Crash box** is a crashbox in a star shape, compressed by an impactor on one side against the ground (see Figure 3a). The variables that are optimized, depending on the configured amount of dimensions, represent the side lengths of the crash box, the depth of the ridges of the star shape, and the thickness of the structure, as demonstrated in Figure 2a. The mathematical formulation consists of a piece-wise function, based on the objective values of *intrusion* ($\delta$) and *specific energy absorbed (SEA)*, and it is defined as follows:

$$\min_x \quad \text{Penalized SEA}(x) = \begin{cases} -\text{SEA}(x) & \delta(x) \leq 60\text{mm} \\ 100(\delta(x) - 60) & \delta(x) > 60\text{mm} \end{cases}$$

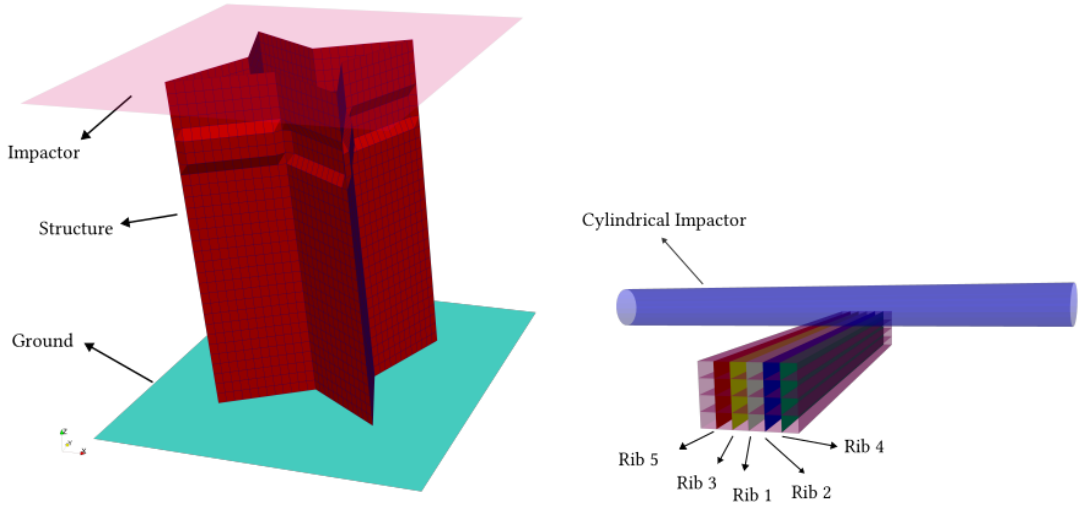
**MECHBench Problem 2 (MB2): Three Point Bending of Layered Beam** is a beam which consists of several separate length-wise ribs, that is fixed at its edges and is bent in the middle by a cylindrical impactor. Each defined dimension here, represents the thickness of a rib (see Figure 2b). Similarly to MB1, the mathematical formulation of this problem is a piece-wise function, this one involving intrusion ($\delta$) and structure mass ($\text{m}_s$), and is defined in the MECHBench paper as:

$$\min_x \quad \text{Penalized } \mathrm{m}_s(x) = \begin{cases} \mathrm{m}_s(x) & \delta(x) \leq 50\text{mm} \\ 4.25952 + 10\left(\frac{\delta}{50} - 1\right) & \delta(x) > 50\text{mm} \end{cases}$$




(a) (b)

Figure 2: Definition of dimensions for MB1 (a) and MB2 (b) (from [32])




(a) Star-Shape Crash Tube (b) Three Point Bending

Figure 3: Visualisation of MB1 and MB2 (from [32])

# 3 Methods & Implementation

## 3.1 MECHBench configuration & adaptation

The two MECHBench problems we use [32], described in Section 2.4.1, are configured for our study as follows:

MB1 (Star-shaped crash box) is configured to five dimensions $d = \{x0, x1, x2, x3, x4\}$, representing side lengths, ridge depth and wall thickness.

MB2 (Three-point bending) is configured to five dimensions as well $d = \{x0, x1, x2, x3, x4\}$, with each dimension representing the thickness of an individual rib of the beam.

Additionally, in order to conduct our designed experiment (see Section 3.5) and apply optimization algorithms, we need to implement for each problem the objective function to be optimized, as defined by the authors of MECHBench[32], but using our trained models instead of the outputs of OpenRadioss. This is simply done by defining a function which evaluates a vector for the relevant objective values of a problem using queries or predictions of the corresponding model, the objective values it outputs are used to construct the return of this function according to the problem definition.

## 3.2 Surrogate Models

Due to the complexity of the landscapes produced by the tested problems, it is difficult to deduce which surrogate model class would fit this application well without testing it. Therefore, for this project, we chose five separate surrogate model classes to test, which represent some of the different core approaches to building such models. The five classes are **Response Surface Methodology**, **Random forest**, **XGBoost**, **Gaussian process** and **Bayesian network**.

In order to accurately model the MECHBench problems, we consider their mathematical formulations (see Section 2.4). For MB1 and MB2, the mathematical formulation consists of a piecewise function, involving both target values of each problem. We replicate this for each model class as follows: For Bayesian network we train a single instance of the model, specifying which nodes created from the labels in the training sets correspond to targets. For the other models, we train an instance for each of the target objective values. This way we model each part of the piece-wise objective function, to obtain a final model objective function on which the optimizer algorithms run.

The models are evaluated on the $R^2$ metric to judge their fitness to the training data. The evaluation done per model instance on each target objective value during the development of the models, and on the assembled modeled objective function to provide a more accurate view of the total performance. The former mentioned evaluation is done during training using a split of the training set, and the latter, is done afther the models are assembled, using a 625 point dataset that for each problem is randomly sampled from the second half of the $500D$ (with $D$ being the amount of dimensions) training set. This is done to ensure that the models are as much as possible tested

on points that are not included in their training set. Specifically, since for both problems, the trainings sets up to $250D$ are created by taking the first $30/60/125/250D$ points of the $500D$ set. This means that the models trained on datasets up to $250D$ are evaluated on points that are not existent in their training sets, and the models trained on $500D$ datasets, are evaluated on 625 points which may exist in the training data, a trade-off made due to time restrictions. The overall $R^2$ scores of each model class at each training set size is provided in Appendix A.

The following subsections outline, for each model class, the basic idea, the reason why it was picked and the tools used to implement it for this experiment. It is important to note that the parameters for the configuration of the models are chosen arbitrarily to be values commonly used for each model, due to time constraints for this project (see Section 6).

#### 3.2.1 Response Surface Methodology (RSM)

RSM encompasses a wide approach to surrogate modeling. For our purpose, we use a Polynomial Response Surface model, which fits the shape of a polynomial function to the points of the training set. In particular, we implemented a quadratic response surface model, which fits the shape of a quadratic function to the extracted polynomial features from our dataset.[12]. For this we used the scikit-learn[31] Python package.

#### 3.2.2 Random Forest

We use Random Forest[36] as a model class to be representative of a bagging ensemble method. For this algorithm, there is a set amount of decision trees, each fit to a random subset of the training set. The predictions of all trees are weighted to produce the final prediction. In our implementation we use the RandomForestRegressor class provided in the scikit-learn[31] Python module, configured to 100 trees, with the rest of the model parameters set to the defaults of the package.

#### 3.2.3 XGBoost

The XGBoost[7] class, is selected to represent and investigate the performance of a boosting ensemble method in this task. It starts with a set amount of decision trees as base learners, the error of which is used to train the next set of decision trees. This process is repeated until the early stopping criterion is reached, meaning for a set amount of rounds the model performance is not improved. At this point the output of all trees of this depth is weighted to produce the final prediction. The configuration used in our implementation is 100 base learners and a max depth of 5 for each tree, which are common starting values, and the stopping process is left to the default of the library used. This configuration however, was chosen due to time constraints and is arbitrary, it would likely benefit from some hyperparameter tuning process, as discussed in the limitations section below (6). For the implementation we use the xgboost Python package available from the authors of the cited work in which the method was proposed.

#### 3.2.4 Gaussian Process

A Gaussian process model[10] is chosen to represent a probabilistic surrogate model class. It works by defining a prior distribution and a kernel. These are then updated with (or fitted to) the

training data, to create a posterior distribution as well as tune the kernel parameters to maximize the likelihood of the data in the training set. The likelihood in our configuration is chosen to the Gaussian likelihood function and the kernel was the package default squared exponential (or radial basis function) kernel. This process requires an optimization loop, for which we use the Adam [23] optimization algorithm. This is implemented using the gpytorch Python library[15].

#### 3.2.5 Bayesian Network

We implement a Bayesian network surrogate model [19] to represent this unique approach to surrogate modeling. This involves learning the structure between features and targets, and how those connections influence each other, and representing this on a Directed Acyclic Graph (DAG). The model then quantifies the conditional effects of nodes to each other using Bayes' rule. To allow for prediction of continuous values, the linear Gaussian Bayesian network[4] directly implemented from the used Python library is utilized. After this process the model is able to be queried with the evidence of some set of values for the features, and make predictions on the values of the targets. For our implementation, we used the hill climb search algorithm[33] to learn the structure from the dataset. This model, including the hill climb algorithm is implemented using the pgmpy Python package[2].

### 3.3 Data Generation & Handling

For the purpose of training the surrogate models, we had to use the MECHBench codebase as provided by the authors to generate the required datasets. In order to investigate the effect of dataset size of these problems on the quality of the results produced by the fitted models, as well as provide insight in regard to the available budget when implementing such a solution, we created and used datasets of 30, 60, 125, 250 and 500 times the dimensionality (D) of each relevant problem. In order to save time on computation, we only generated the $500D$ dataset for each problem, from which we used the first $30D$, $60D$, $125D$, and $250D$ points to create the smaller sets. This was made possible due to the use of Sobol sequence sampling[35] with seeding. That is because this technique provides a similar distribution between smaller and larger samples, and is deterministic, meaning it always generates the same samples in the same order. Therefore, if the same seed is provided for scrambling, a sample of size $30D$, is identical to the first $30D$ points of a sample of size $500D$. We use the Sobol sampling utilities provided by the SciPy[37] Python package to generate $500 \times 5 = 2500$ vectors of five values between $-5$ and 5. These vectors are then each evaluated using the existing code of MECHBench with the OpenRadioss engine[1], to extract the corresponding relevant objective values, which would be used as labels of the training sets.

This process, of course, is very computationally expensive, so it was performed, by running concurrent data generation jobs, on the Academic Leiden Interdisciplinary Cluster Environment (ALICE) High Performance Computing (HPC) facility of Leiden University.

### 3.4 Optimizers

The optimizers we use, both on the evaluations with MECHBench as is and with the surrogate models are the following: **CMA-ES**[16], **TuRBO-1**[13], **BoTorch**[3] and **BAxUS**[30] as implemented

by Santoni et al.[34], **1+1 ES** as implemented by Jacob de Nobel[8] and **Differential Evolution (DE)** as provided by the SciPy library [37]. These algorithms, as with the chosen model classes, represent a range of optimization techniques, and are chosen with the intention of revealing intricacies which may differ between the different configurations of the objective functions of these problems, and in turn result in variation in the algorithm evaluation. This information is useful for our interpretation of the results to answer our RQs, and judge the suitability of our methods for this purpose.

The performance of each optimizer is evaluated on the **Area Over Convergence Curve (AOCC)** metric, adapted from how it is used by Li et al.[28] following the formula below:

$$\mathrm{AOCC}(y_{\alpha,f}) = \frac{1}{B}\sum_{i=1}^{B}\left(1 - \frac{\min(\max(y_{\alpha,f}, lb), ub) - lb}{ub - lb}\right)$$

Where $\alpha$ is the optimization algorithm, $f$ is the objective function, $y$ is the objective value obtained using the objective function and optimizer, $lb$ and $ub$ are the lower and upper bounds, set to the minimum and maximum objective values observed in the convergence data of the optimizer runs of each problem, and B is the budget, set in our case to $30D$. This metric is chosen to base the ranking of the algorithms on, due to the fact that it yields more complete information about the entire optimizer run. This is because its value is influenced by the entire curve up to the point at which it is calculated. On the contrary, simply using the best seen objective value would only reflect the behavior of an optimizer on a landscape on the final evaluation of the run's budget.

## 3.5 Experiment Structure

Considering how our **RQs** are posited, we design an experiment as described below.

For each of the two used MECHBench problems, at the selected dimension configuration, we take the objective function, both the one from the existing MECHBench framework, and the modeled objective function defined based on the fitted surrogate models. This is done for all used training set sizes and for all the model classes. These different versions of the objective functions are wrapped in a logger to track convergence data used for metric calculation. Then they are provided to instantiate and run each of the selected optimization algorithms, which are evaluated on the AOCC metric with a budget of $30 \times D$. This process is repeated 15 times, to account for the stochasticity of the optimization process. The metric values obtained, are used to produce algorithm rankings corresponding to all model classes at all training set sizes, as well as to the performance of the optimizers on MECHBench with the simulation. This is done separately for both problems. The AOCC values and all rankings are interpreted and a comparison for similarity is performed between each ranking on a model and the ranking on MECHBench using Kendall's $\tau$ coefficient[22], to provide an answer for the degree of ranking preservation. The process of our study and the experiment design are visualized in Figure 4.

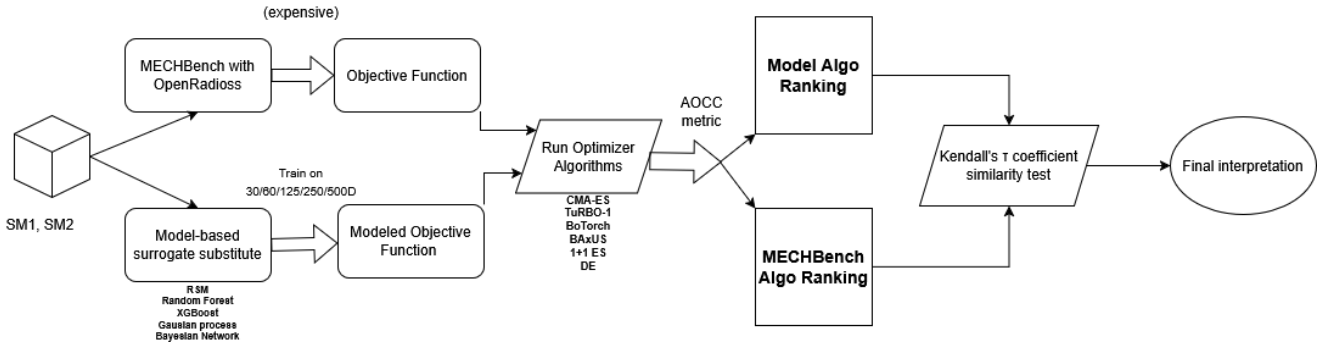



Figure 4: Experiment process structure

## 3.6 Project Flow

The first major component of the project was to implement the different surrogate models. Going over the documentation of the Python packages providing tools for each of the classes, we decided on the specific packages to be used as described in subsection 3.2. Then an iterative process took place, setting up the models and training them on a small 150 point dataset obtained by running MECHBench to evaluate 150 randomized 5 dimension input vectors, and extracting the objective values on which model instances need to be trained to assemble the modeled piecewise objective function for MB1 and MB2. An existent $250D$ dataset was also provided by the supervisors of this thesis for the purpose of testing the models to ensure their implementation is correct. This dataset was created using Latin Hypercube Sampling (LHS)[29] instead of Sobol, so it could not be used in the final training of the surrogate models. During this process, there was no hyperparameter tuning, as is mentioned above, due to the fact that it is outside the scope of this study, however, adjustments were made in how each model is set up, not to maximize performance, but to ensure that the logic followed to model the objective function is sound, and that each model, as well as its results can be comparable to MECHBench.

After the model set up code is ready, we began the process of obtaining the training sets, as described above. This was done by separating batch jobs for each model evaluation to be able to run concurrently on ALICE HPC, this proved to be a challenge, due to the many details that require attention for the generation of a dataset in this manner. Specifically, managing the Python environment to be consistent and compatible with the modules available on the cluster proved to be cumbersome, as well as the fact that at the period of conducting this study the cluster was often overloaded leading to extended wait times for the data generation jobs we submitted to be executed. While this was happening, the code of the model set up was reorganized to a package in cleaner python files, from the interactive python notebooks which were used during the iterative process described for creating the models.

With the datasets acquired, we trained the model instances on a local machine, and started working on adapting the used optimizer algorithm code to fit our experiment set up. Specifically, the preexisting wrappers were used for the CMA-ES, TuRBO-1, BoTorch and BAxUS algorithms taken from Santoni et al.[34], and similar wrappers were implemented for the optimizer functions of DE and 1+1 ES. Additionally, tools offered by the IOH Python interface [9] were used to wrap the

assembled model objective function similarly to how it is done in the code of MECHBench[32], as well as create a logger for both modeled and simulated objective functions to keep track of the optimizer convergence data.

With all the above components completed and set up, we set up jobs to execute the optimizer runs on the 15 seeds on MECHBench and all the trained models on ALICE HPC. The optimizers ran in little time for the modeled objective functions, but took several days when it came to running them on the MECHBench problems. This was the point where the decision was made to exclude the third MECHBench problem (MB3), which until that point, we had set up to investigate in 15 dimensions. This was due to the fact that there was limited time to complete the thesis project, and the needed runs would require, by estimation, at least 100 hours of compute on the cluster, even in the best case, that all 90 optimizer runs corresponding to SM3 (6 algorithms *imes* 15 seeds) would be executed concurrently and immediately, with no wait time to be picked up, which was very unlikely.

While the optimizer runs were executing on the cluster, we prepared to do all the data processing creating for the visualization and analysis of the results. Specifically creating utilities with Python to evaluate the models, calculate the AOCC metric and the Kendall's $\tau$ values, as well as plot the convergence, rankings and all other visualizations shown in Section 4 and the Appendices (A, B).

When the results were obtained, processed and plotted, we performed a presentation and analysis as seen in the Results and Discussion sections (4 and 5 respectively), which lead to our recommendations for further related works (see Section 6) and finally our conclusion (Section 7, addressing our RQs and our Hypotheses.

# 4 Results

After executing the experiment as described above (see section 3), we use the resulting convergence data of the six optimizers on all models, and on MECHBench, across 15 different seeds to present the mean ranking across seeds, and the Kendall's $\tau$ value when comparing the ranking on each model against the ranking on MECHBEnch together with the corresponding $R^2$ performance of each model (See figures below). We also show in violin plots the distances between each best found feature vector by a model, to the best found feature vectors by MECHBench. Additionally, heatmaps of model performance on the $R^2$ metric as well as the convergence curves of the optimizer runs for each model are provided in Appendices A and B respectively.

## 4.1 Algorithm Rankings

The development of algorithm ranking during the evaluations of the optimizer runs based on the AOCC metric, averaged over all 15 seeds, is plotted for all configurations that were ran, and is provided in the figures below. By the definition of AOCC used (described in Section 3.4), the best ranked algorithm (number 1) is the one with the highest AOCC value, and the worst ranked (number 6) is the one with the lowest value.

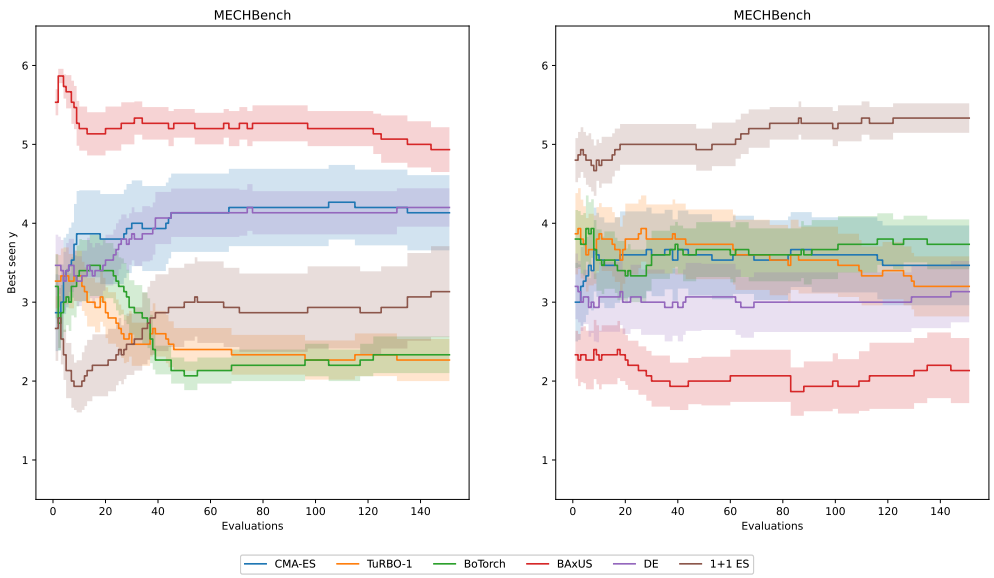



Figure 5: Mean algorithm ranking on MECHBench optimizer runs with AOCC metric (MB1 left, MB2 right)

We provide the mean algorithm ranking plots of the optimizer runs on the MECHBench framework in Figure 5. Additionally plotted in the same manner, are the mean algorithm rankings of the optimizer runs, for each model class on the models of each chosen training set size, seen for MB1 in Figure 6 and for MB2 in Figure 7, each including the corresponding mean ranking on MECHBench at the top for comparison.

## 4.2 Ranking Preservation

The ranking preservation is measured by calculating the value of the Kendall's $\tau$ statistic between the algorithm ranking on a surrogate model and the corresponding ranking on MECHBench. These $\tau$ values are presented in heatmap form in Figure 8, where they are calculated at six slices of the optimizer run budget, as well as in comparison to the $R^2$ score of each model in the scatterplots in Figure 9 for both problems.

## 4.3 Best Vector Distances

As a check of how representative the optima found on the landscape of the modeled objective functions are of the landscape generated by MECHBench, the distances between the vectors corresponding to the best objective values found by each optimizer with the model, and the ones found with MECHBench, are shown in the violin plots in Figure 10.

## 4.4 Clock time per Evaluation

Finally, to assist in the interpretation of the usefulness of our provided solution, the clock time in milliseconds is calculated. This is done by evaluating with each model the same 300 random input

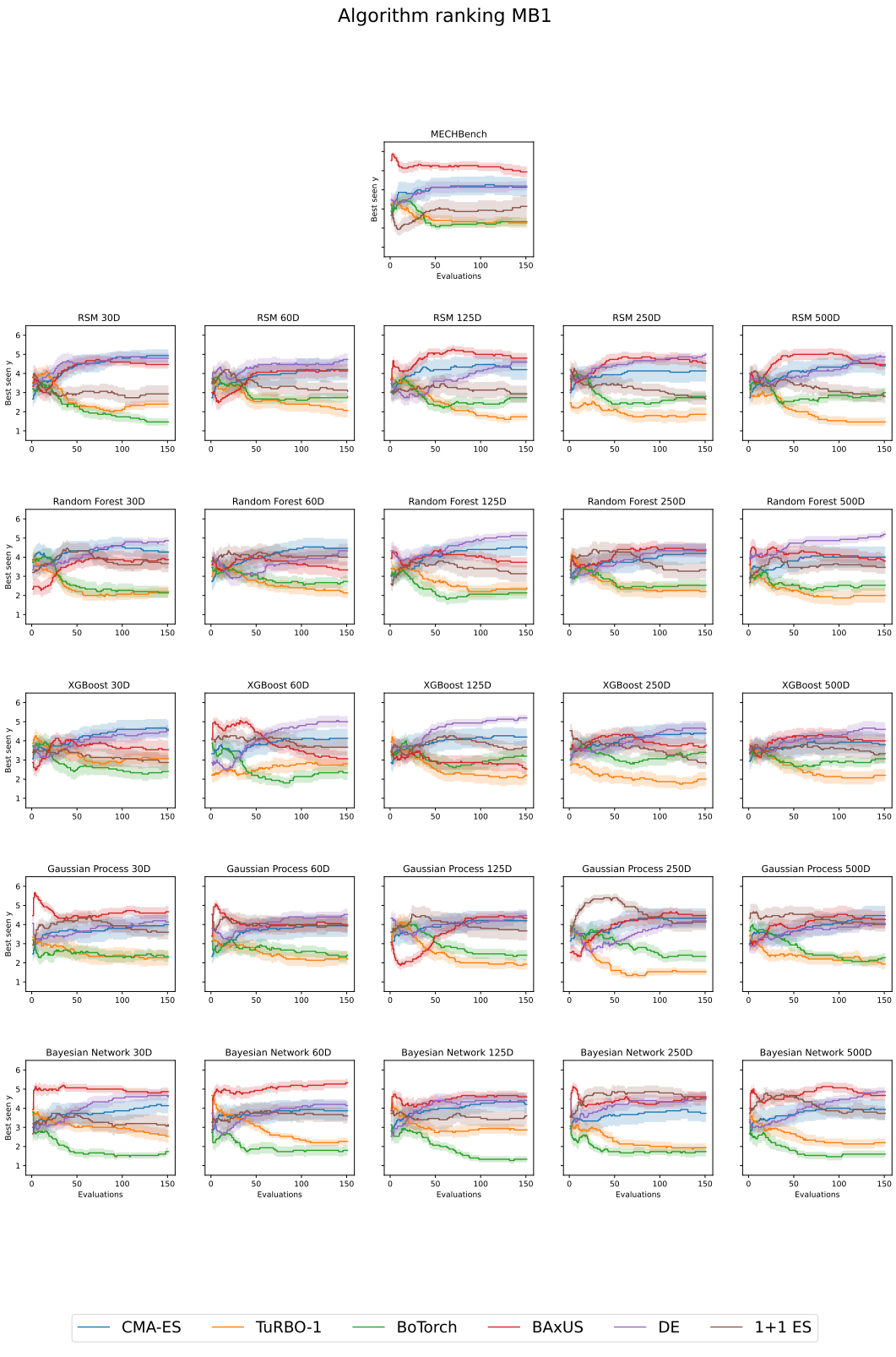



Figure 6: Mean algorithm ranking on surrogate models with AOCC metric (MB1)

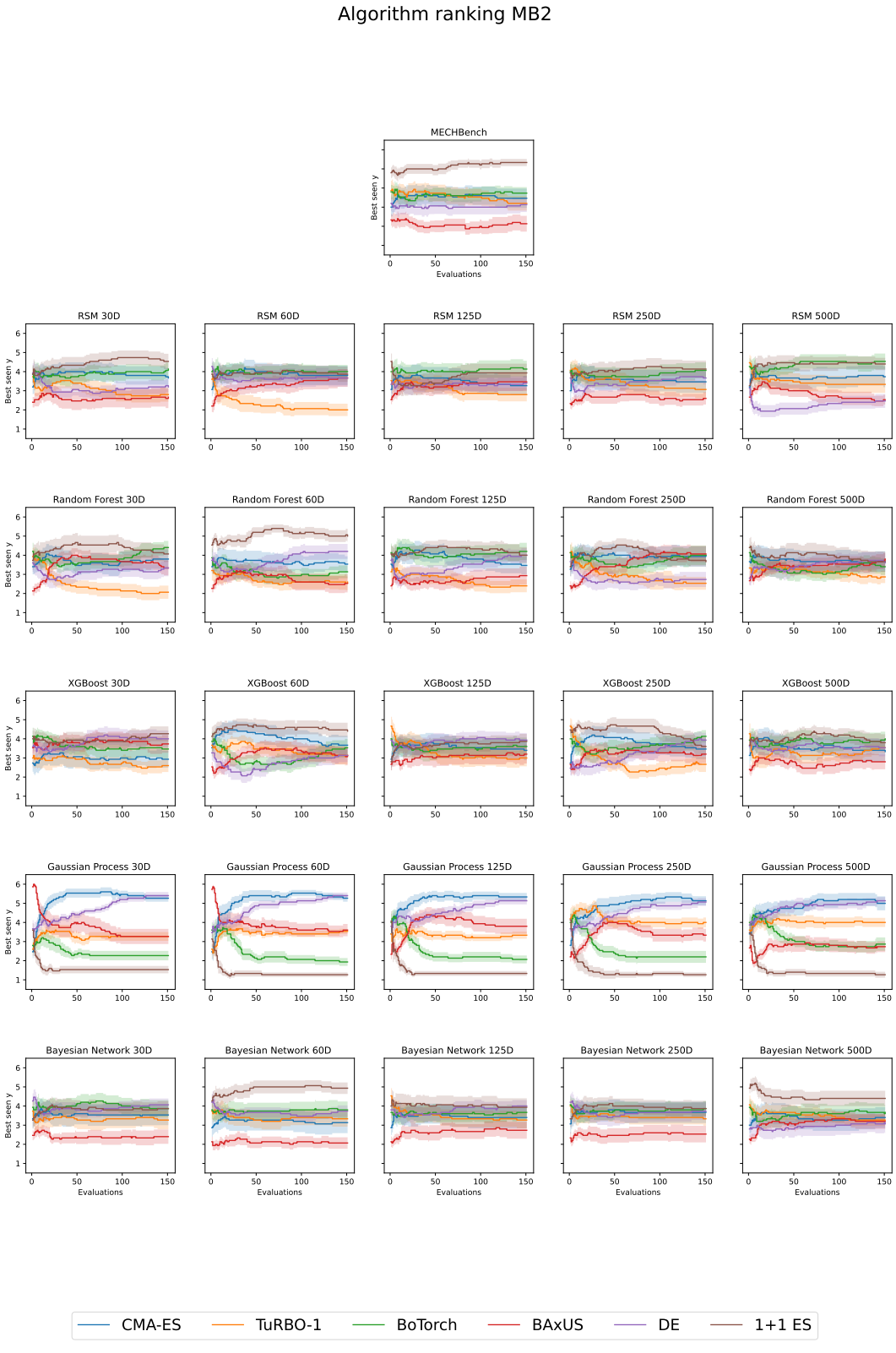



Figure 7: Mean algorithm ranking on surrogate models with AOCC metric (problem 2)

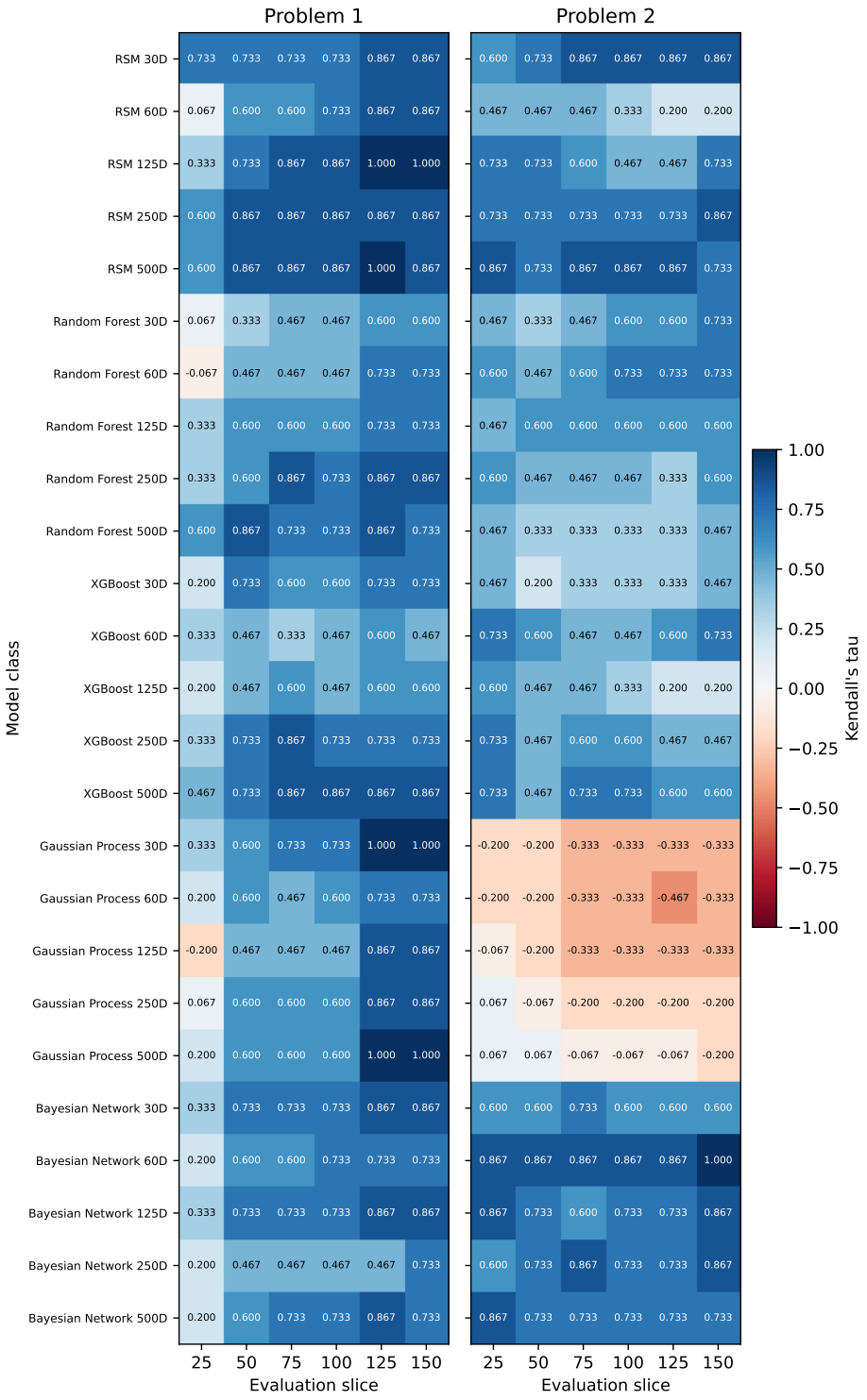



Figure 8: Heatmap of Kendall's $\tau$ values per model for MB1 and MB2

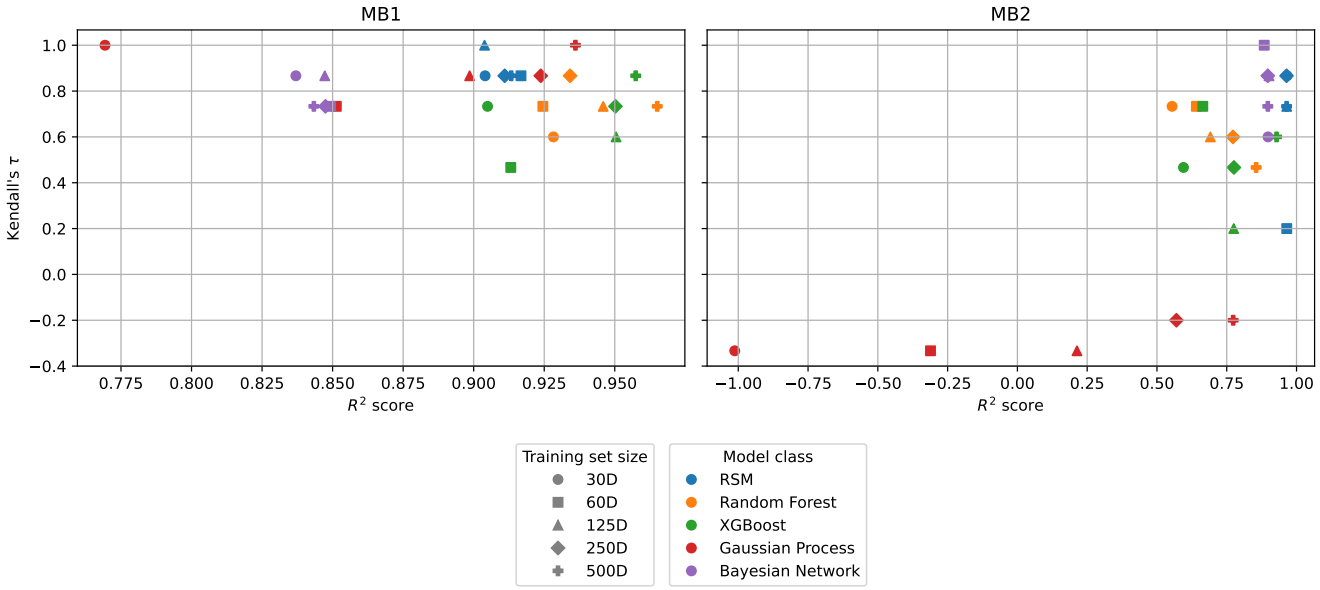



Figure 9: Kendall's $\tau$ values per model compared to $R^2$ scores for MB1 and MB2 based on AOCC metric

vectors and calculating the mean time each model requires to evaluate a vector. This is seen in the heatmaps of Figure 11.

# 5 Discussion

On MB1, the rankings of the TuRBO-1 and BoTorch algorithms followed a similar trajectory throughout the evaluations, and were ranked as the two best performing algorithms, with TuRBO-1 only slightly outperforming BoTorch by the final evaluation. 1+1 ES ranked third, while ranking first during earlier evaluations, during the rest of the run the ranking drops, and by a significant margin. Following, fourth and fifth by only a slight margin of difference were CMA-ES and DE respectively, which also followed particularly similar ranking trajectories throughout evaluations. BAxUS ranked last, and remained last during the entire run, with a dip during early evaluations. When it comes to MB2, BAxUS, contrary to MB1, outranked the other algorithms consistently for the entire run by a considerable margin. DE ranked second, and it also kept that ranking and its distance from the other algorithms consistent during the run. TuRBO-1, CMA-ES, BoTorch were ranked third, fourth and fifth, with similar overall ranking trajectories. Lastly 1+1 ES in this case remained ranked last, becoming lower and lower compared to the rest of the algorithm throughout evaluations.

Looking at the rankings produced by ranking the same algorithms on surrogate classes, of our chosen model classes and training set sizes, there is wide variety in the results. This supports the idea that different model classes behave differently on these systems. In the following subsection, for each model class, observations on the performance and ranking behavior is discussed.

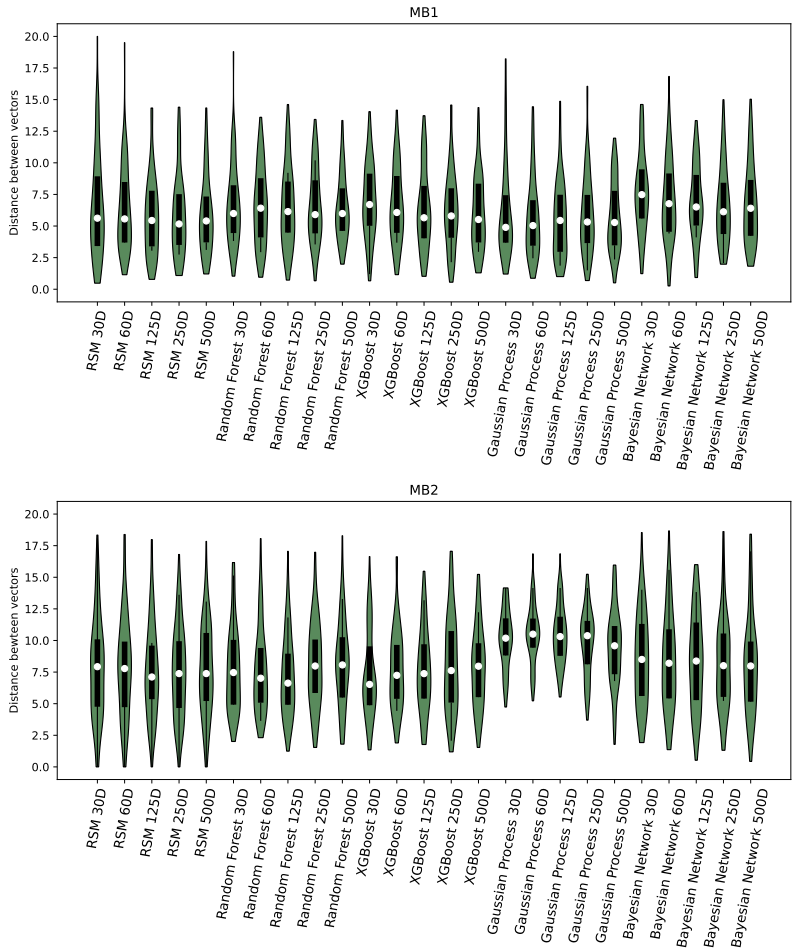



Figure 10: Violin plots of distance of best found vectors of surrogate models from best distance found on MECHBench for problems 1 and 2


Figure 11: Mean time per evaluation (ms)

## 5.1 Per-model Analysis

### 5.1.1 RSM

As seen in Figure 9, the RSM model instances were generally consistent both on $R^2$ scores and on algorithm ranking similarity. On MB1, all instances had similar $R^2$ scores between 0.904 and 0.917, as well as similar degrees of rank preservation with the $125D$ instance fully preserving the ranking with a Kendall's $\tau$ value of 1 and the rest also preserving to a high degree, all with a value of around 0.867. Similarly, in MB2, all models instances produced generally similar scores on the $R^2$ metric with all the assembled instances achieving a value of close to 0.965. There is the an interesting pattern in the Kendall's $\tau$ values which range from 0.2 for the $60D$ instance, while the rest resulted in higher values, around 0.733 for $125D$ and $500D$ and around 0.867 for $30D$ and $250D$. This wide difference in degree of ranking preservation between models with similar $R^2$ scores is curious, but could possibly be attributed to the relatively small budget on which the optimization algorithms ran in our experiment, resulting in different ranking behavior even for models with little differences in how well they are fit to the training sets.

This performance does not make apparent a consistent effect of the size of the training set size on the degree to which ranking is preserved. The high fitness scores as well as the high degree of preservation of the final ranking exhibited by some of the instances partially suggest that such models could have the capacity to be well applicable to these problems, this is also somewhat supported by the fact that, looking at Figures 6 and 16, the optimizer ranking on some of the instances like $125D$ on MB1 as well as $30D$ and $250D$ on MB2, is visibly resembling the behavior of the optimizer ranking on MECHBench during the entire budget of the run.

### 5.1.2 Random Forest

The random forest instances, similarly to RSM, were relatively consistent for the instances corresponding to all training set sizes, with no one being clearly better. On MB1 the model instance preserving the ranking best is the one trained is the one trained on $250D$, with Kendall's $\tau$ being 0.867 and a $R^2$ score of 0.934, and the lowest $\tau$ value being 0.6 for the $30D$ model. All instances seem to have fit well to this function landscape with $R^2$ scores above 0.9. On MB2, there seems to be a trend between the training set size and the accuracy based on the $R^2$, with bigger size corresponding to a higher $R^2$ score, which ranges from 0.554 to 0.855. The inverse trend is seen between training set size and degree of ranking preservation, with $30D$ and $60D$ yielding a Kendall's $\tau$ value of approximately 0.733, $125D$ and $250D$ a lower value of about 0.6 and for the $500D$ instance dropping to about 0.467.

Despite the observed trends on MB2, the trends don't seem to be reflected in MB1. This inconsistency rules out that an actual relation exists, and could be attributed to stochasticity, once again suggesting that there is no overall effect of the training set size on the performance. The degree of preservation generally varies, and when looking at the mean ranking plots in Figures 6 and 7, the behavior of the algorithms throughout the optimizer runs is considerably different between the models and MECHBench, with the run on the $60D$ model showing a relatively similar development of the ranking to MECHBench throughout evaluations.

### 5.1.3 XGboost

In the case of XGBoost, the model instances are placed further away from one another on the scatterplots in Figure 9. A trend which seems to be repeated in both problems, is that higher training set sizes result in a more accurate model based on the $R^2$ metric, with the $125D$ and $250D$ instances scoring almost equally. These $R^2$ are more specifically lie from 0.905 to 0.957 for MB1, and in a wider range for MB2 between 0.595 and 0.929 No trend however is observed in either problem concerning the degree of rank preservation, which varies between instances and only in the case of the $500D$ for MB1 goes over 0.8. This dissimilarity is not only seen in the final achieved rankings, but also in the mean ranking plots throughout the runs when compared to the runs on MECHBench.

These results indicate that an XGBoost model is not consistently able to preserve algorithm ranking, in a set up like the one testing. On the other hand, this does not imply that this model class is necessarily unfit for such tasks, in fact, due to the boosting method used, performing a process of hyperparameter tuning on the model could likely have an effect to improve how well the function landscape is represented by the surrogate model.

### 5.1.4 Gaussian Process

Looking at the results and scores of the Gaussian Process model class when trained on the different training set sizes, we see that in terms of fitness, the $R^2$ score increases with the increase of the size of the training set. In MB1 this range starts from about 0.769 to about 0.936, generally good fitness scores. This is also reflected in the preservation of the algorithm rankings, where the Kendall's $\tau$ value is high across training set sizes, ranging from 0.733 to the highest possible 1 in the cases of $30D$ and $500D$. However, in MB2, while the highest $R^2$ score, achieved by the instance trained on the $500D$ dataset is 0.773, the scores achieved by the models trained on smaller training set sizes get significantly lower, with the lowest one at $30D$ being about $-1.013$, meaning the model predictions are worse than only guessing the mean. In addition, the Kendall's $\tau$ values are below zero for all instances of the gaussian process model class, meaning that not only is the ranking not preserved, but there is a negative correlation between the rankings on the model and on MECHBench.

Overall, this behavior is not unexpected. The model setup and training loop is more complex compared to the rest of the methods, including more sensitive hyperparameters and an entire optimization loop using a gradient optimizer. Considering that no hyperparameter tuning was performed to ensure the best performing model instances possible has been done, the varying performance of the optimizers between model instances and between the two problems is likely justified. Therefore, even though gaussian process seems to generally be the worst out of the model classes tested in our case, we cannot conclude that it is unfit for this task. Our main inference is that gaussian process models without hyperparameter tuning, and especially trained on smaller datasets are likely to be unreliable on such tasks. On that note, the promising results achieved on MB1, suggest that it may be possible to achieve satisfactory performance with such a model, should a tuning process take place for the model set up.

#### 5.1.5 Bayesian Network

The Bayesian network models appear to yield the best results both on MB1 and MB2. There is no visible relation between the training set size and the performance or the value of Kendall's $\tau$. In fact, the $R^2$ scores of all model instances are very close, ranging from 0.837 to 0.849 for MB1 and from 0.884 to 0.901 on MB2. Curiously, despite the same degree of fitness, the different instances exhibit different degrees of preservation of the optimization algorithm rankings with the highest achieved on MB1 being 0.867 by both the $30D$ and $125D$ models, and on MB2 a complete ranking preservation with Kendall's $\tau$ having a value of 1 on the $60D$ model. The variety in ranking preservation despite nearly identical $R^2$ scores, as in the case of the random forest models, can likely be explained as a result of running the optimizers on a relatively low budget ($30D$), which may lead to inconsistent rankings between models with similar accuracy.

Overall, the Bayesian network models, in the way we construct them (see Section 3.2.4) seem to be best fit for this problem even with no hyperparameter tuning. This is further supported by looking at the overall trajectories of the mean algorithm rankings in Figures 15 and 16, which appear to generally be visibly more similar to the corresponding graphs of the optimizer runs on MECHBench throughout all evaluations.

### 5.2 Further Result Interpretation

In the per-model analysis section above, we refer to the ranking plots of Figures 15 and 16 to observe the behavior of the optimizer algorithms throughout all evaluations of the a run. In order to further understand the ranking behavior of the algorithms at different stages in the $30 \times D$ budget, we calculate the values of Kendall's $\tau$ at six equidistant slices of the run as well, to provide information that is not only about the final evaluation. This is seen in the heatmaps of Figure 8. What we observe is that at the first slice (after 25 evaluations), the ranking preservation can have varying values, often very different from the values seen for the same model at the end of the run (after 150 evaluations). This is not unexpected, since at only 25 evaluations, with the stochasticity involved in the optimization process it is likely for the observed algorithm ranking to vary between the surrogate and MECHBench. From the second (after 50 evaluations) to the last slice the Kendall's $\tau$ value changes significantly less for most models, and usually increases or stays the same. There are instances however, like the $125D$ XGBoost model on MB2 where the value starts at about 0.6 in the first slice and drops both at the second, fourth (100 evaluations) and fifth (125 evaluations) slice to about $0.467, 0.333$ and 0.2 respectively. Additionally these heatmaps provide further evidence supporting our observation that training set size does not have a clear effect on the degree to which the surrogate models can preserve algorithm ranking. That is due to the fact that models of the same class, in most cases have similar developments of the Kendall's $\tau$ value, with no clear trend across instances with the different sizes.

Another source of insightful information is given by the violin plots in Figure 10, where the distribution of distances between the vectors corresponding to the best objective value on each surrogate model and on MECHBench. There we observe that for MB1, these distances are mostly distributed between 5 and 7.5 with most models having only a thin distribution of higher distances. For MB2 on the other hand, the distributions are centered around higher distances, mostly between

7.5 and 9, except the Gaussian process instances which are all centered around and somewhat over 10. From this we can infer that between the different classes and MECHBench there seems to be diversity in the optimal solutions present in the function landscape. Specifically, this hints toward the fact that between model classes, landscape information of the original objective function, is preserved to different degrees, leading to the vectors of optimal values found on the surrogate models to have different distances to the corresponding vectors found on MECHBench. Moreover, once again the training set sizes appear to be irrelevant, with the different distributions of each model class having generally similar shapes.

Looking at Figure 11 we see that the average time required by the surrogate models for a single evaluation varies between model classes, and in some cases between training set size as well. Specifically, the RSM and Bayesian network models are the fastest overall and the time stays roughly consistent between training set sizes, with RSM ranging from 0.718 and 1.16 milliseconds on MB1 and from 0.93 to 1.035 milliseconds on MB2, and the Bayesian networks taking from 0.544 to 1.402 milliseconds on MB1 and from 1.568 to 2.698 on MB2. The random forest models also appear to take relatively consistent times around 30 milliseconds in both problems on most training set sizes, with the exception on of the $500D$ instance on MB2 where the evaluation time increases to 42.997 milliseconds. The XGBoost models show a general trend of a relatively higher evaluation time at $30D$ which seems to diminish with the increasing training set size. Finally the Gaussian process models exhibit overall poor performance in terms of evaluation time, where for both problems the evaluation time on the $30D$ instances lies somewhere between the times of the corresponding random forest and XGBoost models, but this generally goes up as training set size increases. Specifically, at the higher sizes it takes significantly longer to make an evaluation compared to the other model classes, especially in the case of $500D$ where a single evaluation takes more than 300 milliseconds (3 seconds). Observing this in the Gaussian process models, it is important to consider that this class was also the only one where in both problems, the model $R^2$ score increased with increasing training set sizes, meaning that the most accurate Gaussian process models also take a much longer time to evaluate an input vector, two whole orders of magnitude longer than the quickest models at $500D$. Nevertheless, all surrogate models classes at all training set sizes are much faster at evaluating a vector compared to the simulation based run of MECHBench, where even at the quickest configuration, using 8 CPU cores the authors report on average 328.5 seconds required for MB1 and about 71.7 seconds required for MB2. This large difference in clock time required between modelled and simulated objective function is consistent with the position held in literature that surrogate models are able to perform many evaluations in little time[20] and reaffirms one of the advantages of using surrogate models, which justifies this investigation, that with such low evaluation times, optimization algorithms and other experiments which require many evaluations of a function can be performed at a fraction of the time required for the same task when using a simulation tool.

## 5.3 Comparison to Related Works

What is found in literature and industry is that there is a general lack of investigation of the applicability of different approaches to surrogate modeling to real-world problems, with few works such as that of Bliek et al. [5]. Related industry solutions often use surrogate model classes that include Gaussian processes, with the default kernels (for example MOPTA-08[21]). Considering

the fact that in our findings, the Gaussian process class appears to be quite inconsistent even at lower dimensionality problems, at least without proper choice of kernel and further hyperparameter tuning, this gap in research seems quite relevant. Furthermore regarding the Bayesian network model class, despite the promising results it exhibits in our study, there does not appear to be many works investigating the applications of this approach to real-world problem optimization.

Our findings also reflect the concerns seen in related literature about surrogate models failing to encapsulate landscape information with increased complexity in the modeled objective function[32][5]. This is specifically hinted towards by the fact that most model instances exhibit lower fitness scores on the $R^2$ metric as well as lower degrees of algorithm ranking preservation, and different ranking behavior throughout optimizer runs, between the relatively simpler function landscape of MB1 and the more complex one of MB2.

## 5.4 Gained Insight

By analyzing the results of our experiment as done in the above sections, we make several inferences which extend our knowledge on this topic. To begin with, the data presents evidence that the size of the datasets used to train the models does not have an effect on how well they preserve the ranking of optimization algorithms. In fact, the biggest difference is consistently made by the different model classes, since on most aspects of the data which we present, overall ranking, Kendall's $\tau$ at different slices, the violin plots of the distance of optimal vectors between models and MECHBench, $R^2$ scores as well as time needed to evaluate input, model class appears to clearly be the distinguishing factor.

In terms of the different model classes, the least amount of insight is gained about the ensemble methods tested, random forest and XGBoost. The performance of these models is neither notably bad or inconsistent, nor promising enough to make specific inferences about their usage in this kind of task, or a recommendation for further investigation. For the Gaussian process model class, what can be inferred is that with no careful selection of configuration and generally some tuning process, the performance both in terms of accuracy and in degree of ranking preservation can be inconsistent and likely unreliable. Additionally, these models, when trained on dataset sizes at which they exhibit high fitness scores, require considerably more time to perform evaluations compared to other models, which could deem them a subpar choice of method in the context of attempting to create surrogate models, on which algorithms can make the many evaluations required for the optimization process with less cost. The RSM models we tested provide promising results at the lowest clock time per evaluation between the models without requiring a larger training set to preserve algorithm ranking better, and could be studied further especially to attempt some tuning process to their setup, to test their applicability in such problems. However, the extent to which this model class could be recommended as fitting to such applications based on these results is limited by the exhibited dissimilarity between the ranking behavior of the algorithms throughout the run, compared to the MECHBench runs. Nevertheless, the model class we more confidently suggest is the best option out of our tested methods is Bayesian networks. Our models, using a linear Gaussian Bayesian network instance, generally provide higher degrees of degree preservation than the RSM models, and also show high $R^2$ scores, this good performance does not only happen for the higher training set sizes. The ranking behavior of the algorithms is

also the most similar on some of the Bayesian network models to the behavior seen in the runs on MECHBench. Additionally, the clock time needed per evaluation is also similarly low for both tested problems. This leads us to recommend their use for such tasks, and seeing the lack of literature testing them in this field, suggest related further investigation, as explained in Section 6.

More generally, from our observations we cannot make conclusive inferences on the preserved optimality or algorithm ranking between the modeled and original objective functions of MB1 and MB2. In fact, the observed diversity in optima present in the modeled objective functions, between the different model classes, suggests that the optimality of the original function landscape is not necessarily preserved by the surrogate models, this is hinted towards but is still inconclusive based on our findings and would require a different experiment setup. Regarding the ranking specifically, despite the promising findings in our results, there is no clear suggestion that ranking is generally preserved, nor that it is not, or cannot be, when using surrogate models for real-world optimization tasks. The findings described above, and the inferences described in this subsection are used to formulate the final addressing of our RQs and hypotheses in Section 7.

# 6 Limitations & Future Work

The main limitation on the scope of this study sources from the constrained available time to complete this thesis work, as well as the limited computing time we had access to on ALICE HPC. This means that we were unable to generate training datasets on multiple seeds to incorporate validation folds in the training of the models, as well as be able to study the variability on performance of each model class, on different seed datasets of the same size. Furthermore, with more available time and resources, we would have been able to perform hyperparameter tuning on the parameters of the surrogate models, which would allow us to ensure that the performance measured in the results and used for interpretation is that of the best possible model configuration. Another point that could benefit from further analysis which fell, as well due to time constraints, outside of the scope of this thesis study, was a further interpretation of the performance of each optimization algorithm, related to its design. This may be a source of insight on where, how and why some surrogate models may perform well or worse. Most importantly, these constraints made it impossible to include the testing of the third MECHBench problem at higher dimensionality (15 dimensions), which would allow us to make more inferences regarding a more complex system, and test our surrogate model approaches against the position in literature that such solutions are weak when dimensionality is scaled.

For future works, the points mentioned in the above paragraph can all be incorporated in a replicating or expanding study based on the MECHBench framework. Furthermore, the models classes which showed promising results in our experiment, especially Bayesian networks (specifically linear Gaussian Bayesian networks) but also Random forest should be tested further on real-world optimization problems, to investigate their applicability, as a cheap solution to represent these systems. It is also worthwhile to investigate, which configuration choices (kernel, optimizer, etc) can lead to a Gaussian process based model which performs best on such tasks, since the capacity to model an objective function is hinted to have potential, through the results we obtained when running such a model with no tuning or carefully selected configuration on MB1. A consideration for these studies would be to increase the budget on which the optimization algorithms are ran,

since in our results we ran into points of curiosity that are likely attributed to the low budget. Lastly, a study incorporating these proposals, could provide wider benchmarking insight, by testing these model classes with our provided code framework in a similar experiment design and including the problem functions in BBOB suite proposed by Hansen et al.[18], this is convenient because these exist in IOHExperimenter[9], therefore requiring minimal adaptation of the code framework we developed for this study.

# 7 Conclusion

Finally, to address our RQs and hypotheses, we combine our observations and analysis presented and discussed in sections 4 and 5 and based on these formulate the conclusions described below.

Starting from our main RQ, where we asked whether optimization algorithm ranking is preserved between our model-based solution and MECHBench, and hypothesized that it is not, the findings seem to be inconclusive. The Bayesian network models, in both tested problems, generally showed high degrees of algorithm presentation, at high $R^2$ scores, with some of the instances leading to similar behavior of the algorithms between the model and MECHBench throughout all evaluations of the run. However, this was not true of all the Bayesian network instances. The rest of the model classes, also exhibit instances which achieve high or even full preservation of ranking, but show inconsistency between MB1 and MB2, or dissimilarity to MECHBench in the ranking behavior of the algorithms during the runs. The most noteworthy inconsistency being the very promising degree of ranking preservation of the Gaussian process instances on MB1, compared to the negative correlation of ranking observed from these models on MB2. Despite the inconsistency and the observed poor performance, considering the amount of promising results found across model classes (but mainly on Bayesian network) as well as the fact that no hyperparameter tuning was performed to ensure the best possible model instances were obtained, we believe that it cannot be rejected that surrogate models are capable of preserving algorithm ranking in these problems. This is why, as mentioned in Section 6, further testing, which will include hyperparameter tuning, is proposed, to be able to conclusively answer this RQ.

Concerning our subRQ and the corresponding hypothesis, where we asked what effect the training set size has ranking preservation, and hypothesized that there would be a linear relation between training set size and algorithm ranking preservation, we did not observe any effect. In fact, judging by the observations in comparing final degree of preservation to the $R^2$ of a model (Figure 9), the ranking throughout evaluations (6 7), the evolution of the value of Kendall's $\tau$ in different stages of the optimizer runs and the distribution of distances of the vectors corresponding to optima found by the models and by MECHBench (10), the size of the training set of the models does not appear to be relevant. Some relations are observed, but are either only regarding the $R^2$ score, or inconsistent between MB1, and MB2, so therefore likely attributed to stochasticity. The fact that our hypothesis is rejected is valuable knowledge in the context of seeking cheap alternatives to EFEM simulations for real-world problems, because it means that larger datasets, which are computationally expensive to obtain for training, are not necessary to create a model which exhibits the highest potential of the corresponding model class.

Overall, through the interpretation of this data we have gained valuable insight into the use of surrogate models for real-world optimization tasks in structural mechanics, both regarding how we answer our RQs, but also beyond just that. This includes comparisons of the strengths and weaknesses of each model class, solid evidence that the size of a model's traning set is not relevant to the degree of algorithm ranking preservation, and despite the findings being inconclusive as to whether ranking is generally preserved by using surrogate models or not, the results allow us to be able to recommend the use of Bayesian network models for set ups similar to our tests as well as to provide an important stepping stone for the proposed further research into this topic, in the process of determining computationally cheap alternatives to EFEM simulations in real-world optimization problems.

# References


[1] Altair and Contributors. Openradioss, 2025. https://github.com/OpenRadioss/OpenRadioss.

[2] Ankur Ankan and Johannes Textor. pgmpy: A Python Toolkit for Bayesian Networks. Technical report, 2024.

[3] Maximilian Balandat, Brian Karrer, Daniel R. Jiang, Samuel Daulton, Benjamin Letham, Andrew Gordon Wilson, and Eytan Bakshy. BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. *Advances in Neural Information Processing Systems*, 2020-December, 10 2019.

[4] Arnab Bhattcharyya, Davin Choo, Rishikesh Gajjala, Sutanu Gayen, and Yuhao Wang. Learning Sparse Fixed-Structure Gaussian Bayesian Networks. Technical report.

[5] Laurens Bliek, Arthur Guijt, Rickard Karlsson, Sicco Verwer, and Mathijs de Weerdt. Benchmarking surrogate-based optimisation algorithms on expensive black-box functions. *Applied Soft Computing*, 147, 11 2023.

[6] Paul Du Bois, Clifford C Chou, Bahig B Fileta, Tawfik B Khalil, Albert I King, Hikmat F Mahmood, Harold J Mertz, Jac Wismans, Priya Prasad, and Jamel E Belwafa. VEHICLE CRASHWORTHINESS AND OCCUPANT PROTECTION Automotive Applications Committee American Iron and Steel Institute Southfield, Michigan. 2004.

[7] Tianqi Chen and Carlos Guestrin. XGBoost: A Scalable Tree Boosting System. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 13-17-August-2016:785–794, 3 2016.

[8] Jacob de Nobel. algorithms. https://github.com/jacobdenobel/algorithms.

[9] Jacob de Nobel, Furong Ye, Diederick Vermetten, Hao Wang, Carola Doerr, and Thomas Bäck. IOHexperimenter: Benchmarking Platform for Iterative Optimization Heuristics. *Evolutionary computation*, 32(3):205–210, 11 2021.

[10] Thomas Dietterich, Christopher Bishop, David Heckerman, Michael Jordan, and Michael Kearns. Gaussian Processes for Machine Learning Adaptive Computation and Machine Learning.

[11] Fabian Duddeck. Multidisciplinary optimization of car bodies. *Structural and Multidisciplinary Optimization*, 35(4):375–389, 4 2008.

[12] Jeffrey R Edwards. Alternatives to difference scores: Polynomial regression analysis and response surface methodology. 2002.

[13] David Eriksson, Michael Pearce, Jacob Gardner, Ryan D Turner, and Matthias Poloczek. Scalable Global Optimization via Local Bayesian Optimization. In H Wallach, H Larochelle, A Beygelzimer, F d Alché-Buc, E Fox, and R Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[14] Jianguang Fang, Guangyong Sun, Na Qiu, Nam H. Kim, and Qing Li. On design optimization for structural crashworthiness and its state of the art. *Structural and Multidisciplinary Optimization*, 55(3):1091–1119, 3 2017.

[15] Jacob R. Gardner, Geoff Pleiss, David Bindel, Kilian Q. Weinberger, and Andrew Gordon Wilson. GPyTorch: Blackbox Matrix-Matrix Gaussian Process Inference with GPU Acceleration. *Advances in Neural Information Processing Systems*, 2018-December:7576–7586, 9 2018.

[16] N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 312–317. IEEE.

[17] Nikolaus Hansen, Anne Auger, Raymond Ros, Olaf Mersmann, Tea Tušar, and Dimo Brockhoff. Coco: a platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36(1):114–144, 2021.

[18] Nikolaus Hansen, Steffen Finck, Raymond Ros, and Anne Auger. Real-Parameter Black-Box Optimization Bench-marking 2009: Noiseless Functions Definitions.

[19] David Heckerman. A tutorial on learning with bayesian networks, 2020.

[20] Chun Kit Jeffery Hou and Kamran Behdinan. Dimensionality Reduction in Surrogate Modeling: A Review of Combined Methods. *Data Science and Engineering*, 7(4):402–427, 12 2022.

[21] Donald R. Jones. Large-scale multi-disciplinary mass optimization in the auto industry. Technical report, MOPTA 2008, Detroit, MI, USA, 2008.

[22] M. G. KENDALL. A NEW MEASURE OF RANK CORRELATION. *Biometrika*, 30(1-2):81–93, 6 1938.

[23] Diederik P. Kingma and Jimmy Lei Ba. Adam: A Method for Stochastic Optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 12 2014.

[24] Takehisa Kohira, Oyama Akira, Hiromasa Kemmotsu, and Tomoaki Tatsukawa. Proposal of benchmark problem based on real-world car structure design optimization*. *GECCO 2018 Companion - Proceedings of the 2018 Genetic and Evolutionary Computation Conference Companion*, pages 183–184, 7 2018.

[25] Jakub Kudela. A critical problem in benchmarking and analysis of evolutionary computation methods. *Nature Machine Intelligence*, 4(12):1238–1245, 12 2022.

[26] Gawel Kus and Miguel A. Bessa. Gradient-free neural topology optimization: Towards effective fracture-resistant designs. 9 2024.

[27] Jiaqi Li, Jianglong Liu, Muwei Shu, and Jiaxuan Zheng. Finite element and physics-informed neural network fusion for elasticity mechanics solving techniques. *https://doi.org/10.1117/12.3085008*, 13953:235, 11 2025.

[28] Wenhu Li, Niki van Stein, Thomas Bäck, and Elena Raponi. LLaMEA-BO: A Large Language Model Evolutionary Algorithm for Automatically Generating Bayesian Optimization Algorithms. 5 2025.

[29] M. D. McKay, R. J. Beckman, and W. J. Conover. A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. *Technometrics*, 21(2):239, 5 1979.

[30] Leonard Papenmeier, Luigi Nardi, and Matthias Poloczek. Increasing the scope as you learn: Adaptive bayesian optimization in nested subspaces. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[31] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[32] Iván Olarte Rodríguez, Maria Laura Santoni, Fabian Duddeck, Carola Doerr, Thomas Bäck, and Elena Raponi. MECHBench: A Set of Black-Box Optimization Benchmarks originated from Structural Mechanics. 11 2025.

[33] Stuart Jonathan Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, England, third edition, 2016.

[34] Maria Laura Santoni, Renato De Leone, Elena Raponi, and Carola 2024 Doerr. Comparison of High-Dimensional Bayesian Optimization Algorithms on BBOB. 1, 6 2024.

[35] I. M. Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1 1967.

[36] Tin Kam Ho. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, pages 278–282. IEEE Comput. Soc. Press.

[37] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R.J. Nelson, Eric Jones, Robert Kern, Eric Larson, C. J. Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, Aditya Vijaykumar, Alessandro Pietro Bardelli, Alex Rothberg, Andreas Hilboll, Andreas Kloeckner, Anthony Scopatz, Antony Lee, Ariel Rokem, C. Nathan Woods, Chad Fulton, Charles Masson, Christian Häggström, Clark Fitzgerald, David A. Nicholson, David R. Hagen, Dmitrii V. Pasechnik, Emanuele Olivetti, Eric Martin, Eric Wieser, Fabrice Silva, Felix Lenders, Florian Wilhelm, G. Young, Gavin A. Price, Gert Ludwig Ingold, Gregory E. Allen, Gregory R. Lee, Hervé Audren, Irvin Probst, Jörg P. Dietrich, Jacob Silterra, James T. Webber, Janko Slavič, Joel Nothman, Johannes Buchner, Johannes Kulick, Johannes L. Schönberger, José Vinícius de Miranda Cardoso, Joscha Reimer, Joseph Harrington, Juan Luis Cano Rodríguez, Juan Nunez-Iglesias, Justin Kuczynski, Kevin Tritz, Martin Thoma, Matthew Newville, Matthias Kümmerer, Maximilian Bolingbroke, Michael Tartre, Mikhail Pak, Nathaniel J. Smith, Nikolai Nowaczyk, Nikolay Shebanov, Oleksandr Pavlyk, Per A. Brodtkorb, Perry Lee, Robert T. McGibbon, Roman Feldbauer, Sam Lewis, Sam Tygier, Scott Sievert, Sebastiano Vigna, Stefan Peterson, Surhud More, Tadeusz Pudlik, Takuya Oshima, Thomas J. Pingel, Thomas P. Robitaille, Thomas Spura, Thouis R. Jones, Tim Cera, Tim Leslie, Tiziano Zito, Tom Krauss, Utkarsh Upadhyay, Yaroslav O. Halchenko, and Yoshiki Vázquez-Baeza. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nature Methods 2020 17:3*, 17(3):261–272, 2 2020.

[38] Nicholas Zabaras, Shankar Ganapathysubramanian, and Qing Li. A continuum sensitivity method for the design of multi-stage metal forming processes. *International Journal of Mechanical Sciences*, 45(2):325–358, 2 2003.

[39] Martin Zaefferer and Frederik Rehbach. Continuous Optimization Benchmarks by Simulation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12269 LNCS:273–286, 8 2020.

# Appendices

## A Trained Model $R^2$ Scores

In the heatmaps below, the performance of all surrogate models on the $R^2$ metric is outlined, calculated as described in Section 3.

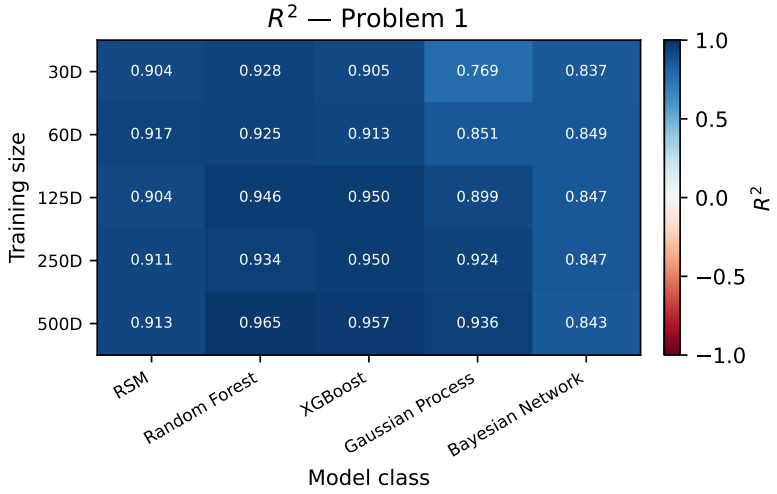



Figure 12: $R^2$ scores per model class/training set size - MB1

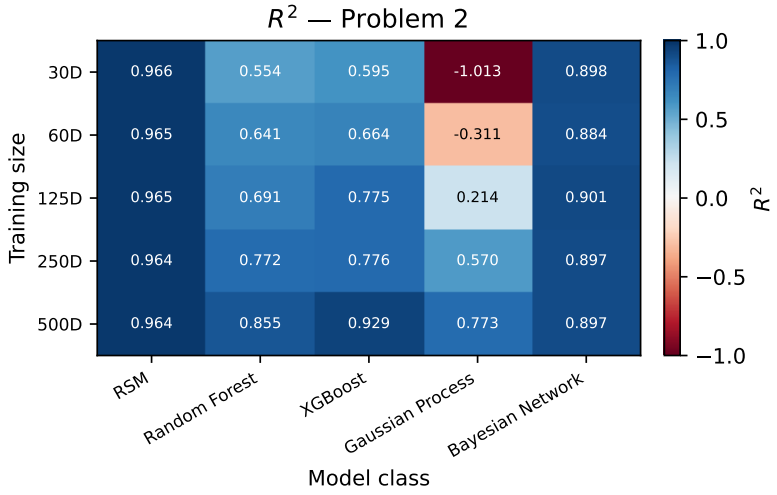



Figure 13: $R^2$ scores per model class/training set size - MB2

# B Convergence Curves

In the following figures, the convergence curves of the optimizer runs are plotted. Figure 14 shows the convergence curve of the optimization algorithms on MECHBench, on the original function landscapes of MB1 and MB2, and Figures 15 and 16 show the convergence curves of the optimizer runs on each surrogate model, for MB1 and MB2 respectively.

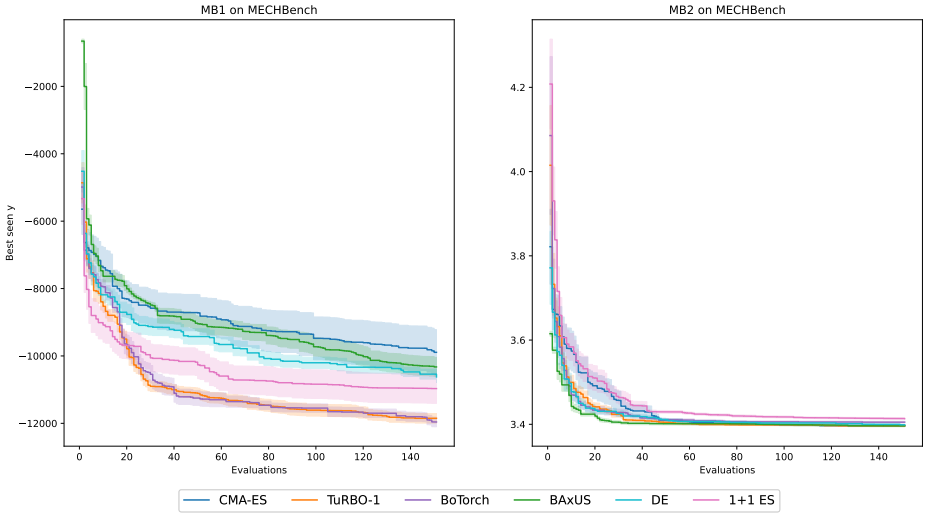



Figure 14: Mean convergence curves of algorithms on MECHBench

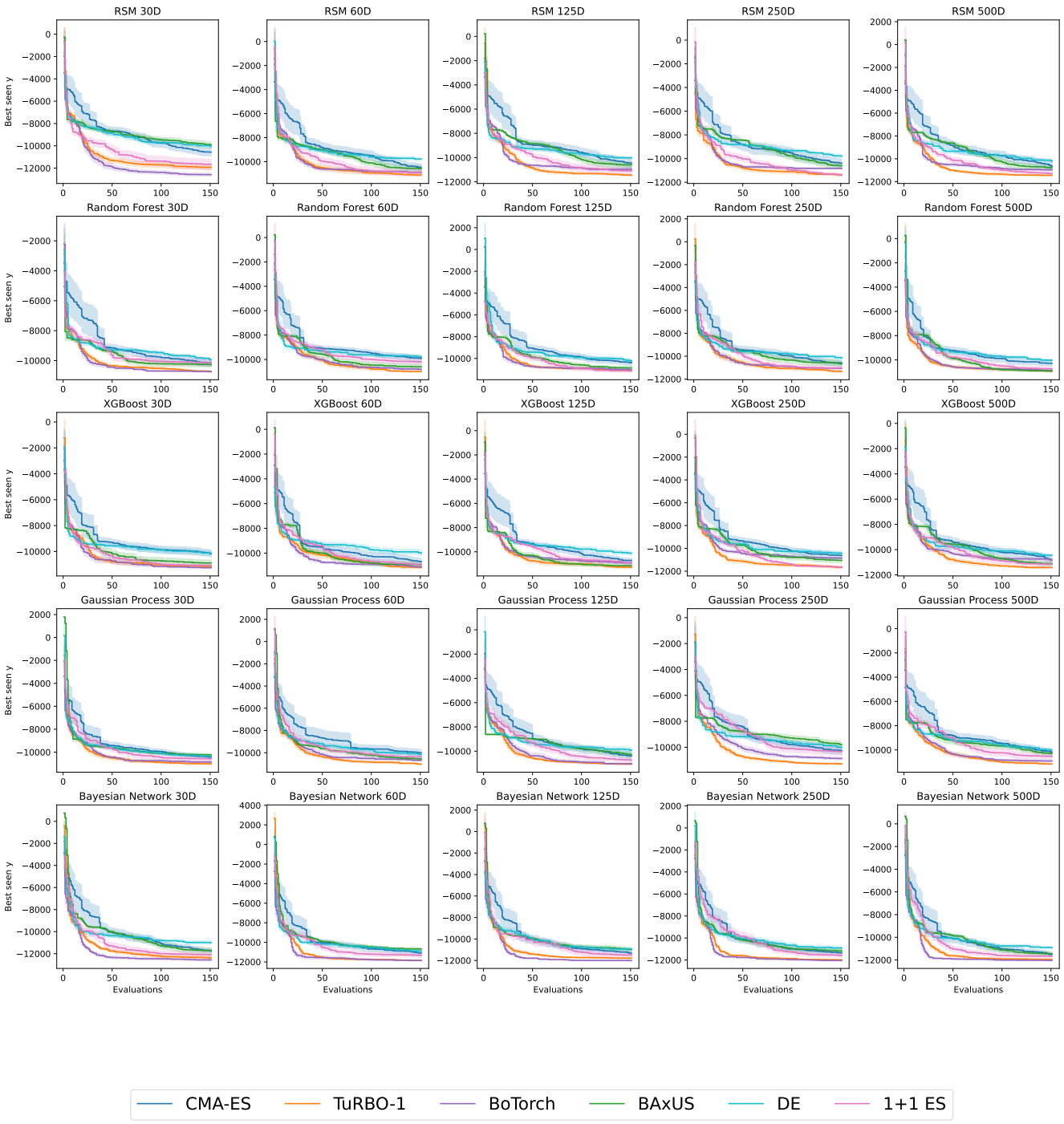



Figure 15: Mean convergence curves of algorithms on surrogate models (MB1)

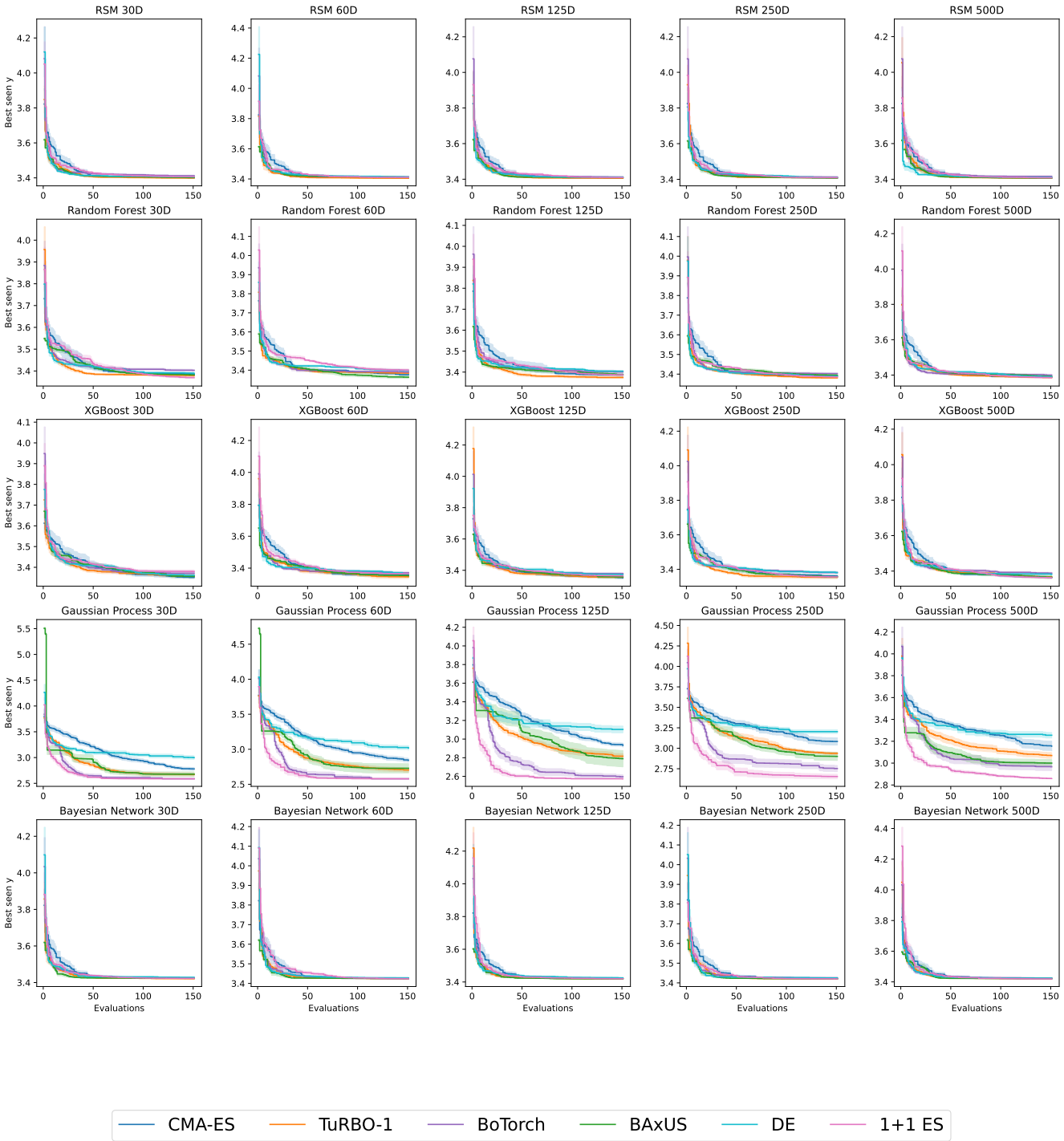



Figure 16: Mean convergence curves of algorithms on surrogate models (MB2)