Evaluation of Neural Architecture Search Approaches for Offshore Platform Offset Prediction

Tomaz M. Suller[1] **, Eric O. Gomes**[1] **, Henrique B. Oliveira**[1] **,**
Lucas P. Cotrim[1] **, Amir M. Sa'ad**[1] **, Ismael H. F. Santos**[2] **, Rodrigo A. Barreira**[2] **,**
Eduardo A. Tannuri[1] **, Edson S. Gomi**[1] **, Anna H. R. Costa**[1]

[1]Escola Politécnica da Universidade de São Paulo (USP)
Av. Prof. Luciano Gualberto, 380, 05508-010, Sao Paulo-SP, Brazil

[2]Petrobras, Rio de Janeiro-RJ, Brazil

{tomaz.suller,ericog,henrique.barrosoliveira}@usp.br
{lucas.cotrim,amir.saad,eduat,gomi,anna.reali}@usp.br
{ismaelh,barreira}@petrobras.com.br

Abstract. *This paper proposes a solution based on Multi-Layer Perceptron (MLP) to predict the offset of the center of gravity of an offshore platform. It also performs a comparative study with three optimization algorithms – Random Search, Simulated Annealing, and Bayesian Optimization (BO) – to find the best MLP architecture. Although BO obtained the best architecture in the shortest time, ablation studies developed in this paper with hyperparameters of the optimization process showed that the result is sensitive to them and deserves attention in the Neural Architecture Search process.*

1. Introduction

The current design process of mooring systems for Floating Oil Production and Offloading units (FPSOs) is highly dependent on the availability of mathematical models and accuracy of dynamic simulations, through which resulting time series motion is evaluated according to platform design constraints. The maximum distance of the FPSO's center of gravity (CG) to its equilibrium position, also known as *offset*, is a critical variable in mooring system design that is dependent on mooring line configuration and incident environmental conditions. The usual approach to determining the FPSO offset is based on dynamic simulations, which tend to be computationally expensive and only allow the assessment of a narrow subset of the possible environmental conditions to which the platform could be subjected. Furthermore, multivariate and complex models of motion dynamics are often based on simplifying assumptions and, therefore, prone to error.

The versatility of neural networks (NN) and deep learning (DL) have allowed *data-driven motion estimators* to be developed. The use of NN decouples the calculation of the CG offset from manually derived models and from dynamic simulations, reducing the computational cost associated with evaluating each environmental condition and, therefore, allowing for the creation of a more holistic picture of the operating conditions of the platform. Previous studies have highlighted the benefits of using NNs for FPSO motion prediction. [Gumley et al. 2016] successfully implemented a Multi-Layer Perceptron (MLP) NN capable of predicting the hourly mean offset of a turret-moored FPSO from environmental conditions. [Mazaheri et al. 2003] also proposed an MLP for predicting FPSO motions such as surge, sway and offset from a set of environmental conditions, showing that it produced results comparable to those calculated by a mathematical model. Despite their promising results, these systems still pose the challenge of determining the best hyperparameter and architecture configurations of the network. Indeed, there is often a reliance on manual, painstaking experimentation and tuning during NN engineering.

The field of Neural Architecture Search (NAS) studies the automation of NN architecture engineering, which has been shown to provide exceptional results that outperform manually designed architectures in certain problems, such as image classification and object detection [Zoph et al. 2017] and regression [Märtens and Izzo 2019]. Research in NAS has increased significantly over the years [Liu et al. 2018, Jin et al. 2018], and has given rise to numerous new strategies [Pinos et al. 2021, Zoph and Le 2016], bringing up a new problem, which is choosing the most appropriate NAS technique.

Therefore, the contributions of this paper are two-fold: (i) a proposal based on MLP to solve the FPSO offset prediction problem; and (ii) a comparative analysis of three different MLP hyperparameter optimization algorithms – Random Search, Simulated Annealing [van Laarhoven and Aarts 1987], and Bayesian Optimization [Frazier 2018]. Random Search is a NAS approach often employed as a baseline to attest the efficiency of NAS algorithms, yet has been reported to match the performance of more advanced techniques [Sciuto et al. 2019]; Simulated Annealing is a well-known and easy to implement local optimization algorithm which has shown promising results [Goffe et al. 1994]; and Bayesian Optimization is a state-of-the-art global optimization algorithm which has seen wide adoption in NAS [White et al. 2019].

The remainder of this paper is organized as follows: Section 2 describes CG offset prediction in FPSOs and the proposed solution. Section 3 gives an overview of NAS and of the optimization techniques explored. Section 4 presents the experiments performed and their respective results, and Section 5 concludes discussing the results of the comparative analysis of the hyperparameter optimization algorithms tested and indicating possible future work.

2. FPSO Offset Prediction

An MLP model capable of predicting maximum platform offset from incident environmental conditions is proposed. This approach bypasses the need for traditional time series dynamic simulation, as the trained model takes in ten inputs corresponding to measured oceanic conditions and directly calculates two outputs corresponding to the desired motion statistics. The proposed model is represented in Figure 1 [Cotrim et al. 2021].

The NN is trained using real current, wind and wave data measured in 3h periods at the Campos Basin in Rio de Janeiro, Brazil, from 2003 to 2010. The measured data consists of: **Current velocity**: Mean current velocity v_c (m/s); **Current direction**: Current propagation angle θ_c (°); **Wind velocity**: Mean wind velocity v_w (m/s); **Wind direction**: Wind incidence angle θ_w (°); **First Wave component height**: Significant wave height H_{s1} (m) corresponding to highest energy wave; **First Wave component period**: Peak Period T_{p1} (s) corresponding to highest energy wave; **First Wave component direction**: Incidence angle θ_1 (°) corresponding to highest energy wave; **Second Wave component height**: Significant wave height H_{s2} (m) corresponding to second highest energy wave; **Second Wave component period**: Peak Period T_{p2} (s) corresponding to second highest energy wave; **Second Wave component direction**: Incidence angle θ_2 (°) corresponding

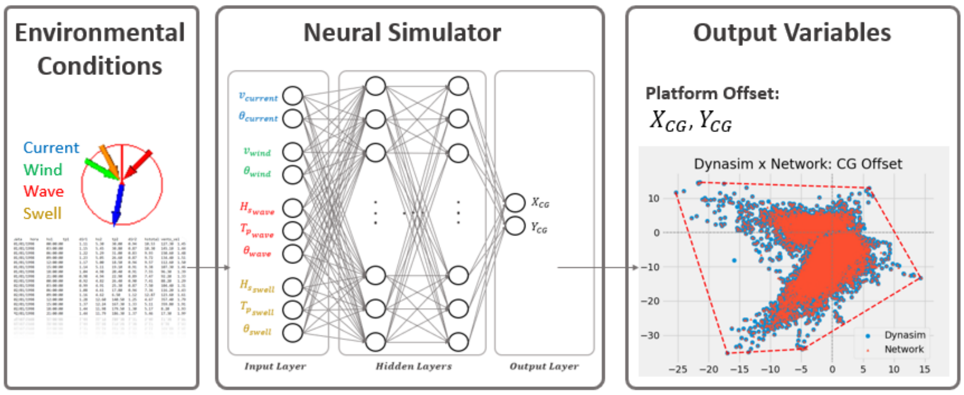


Figure 1. Proposed model for offshore platform offset prediction.

to second highest energy wave.

Subject to these conditions, time-domain simulations using Dynasim [Nishimoto et al. 2002] for a spread-moored FPSO is performed. The time series generated by Dynasim is then analyzed in order to extract X_{CG} and Y_{CG}, corresponding to the global positions of the platform's CG associated with maximum observed offset.

$$\begin{cases} X_{CG} &= X(t^*) \\ Y_{CG} &= Y(t^*) \end{cases} \quad \text{where } t^* = \underset{t>t_{cutoff}}{\arg\max} \left\| (X(t), Y(t)) - (X_{eq}, Y_{eq}) \right\|_2 .$$

During the first seconds the resulting motion is highly dependent on the initial configuration, while subsequent dynamics are governed by the incident environmental conditions. In order to isolate the effects of environmental conditions, a cutoff time $t_{cutoff} = 2000$ s was implemented and time-series analysis was performed from this time onward.

Processing input data. Let $\mathbf{e} = (v_c, \theta_c, v_w, \theta_w, H_{s1}, T_{p1}, \theta_1, H_{s2}, T_{p2}, \theta_2)$ be a set of observed environmental conditions as defined previously. As angular variables are defined in $[0°, 360°]$, their periodic property implies that values such as $0.1°$ and $359.9°$ are functionally close despite being numerically distant. This can cause slow NN convergence as similar environmental conditions may be far apart in the network input space. As a result, the projections of current velocity, wind velocity and wave height in the N-S and E-W directions were used, rather than their magnitude and incidence angle, so that the same set of environmental conditions can be represented as

$$\mathbf{e}^{\mathbf{proj}} = (v_c \sin(\theta_c), v_c \cos(\theta_c), v_w \sin(\theta_w), v_w \cos(\theta_w),$$
$$H_{s1} \sin(\theta_1), H_{s1} \cos(\theta_1), T_{p1}, H_{s2} \sin(\theta_2), H_{s2} \cos(\theta_2), T_{p2}).$$

Each input was then standardized to ensure zero-mean and unit-variance, and the resulting dataset was then divided into train, validation and test sets, as illustrated in Figure 2, using a 5-fold cross validation split for the train and validation sets.

Designing the MLP model. The use of an MLP is proposed here for its ability to exhibit non-linear behavior. In addition to the input and output layers, an MLP is a dense, fully connected network that can have multiple hidden layers. Thus, two decisions must be made: how many hidden layers to have in the MLP and how many neurons there are in each of those layers. With complex datasets involving time series, two or more hidden

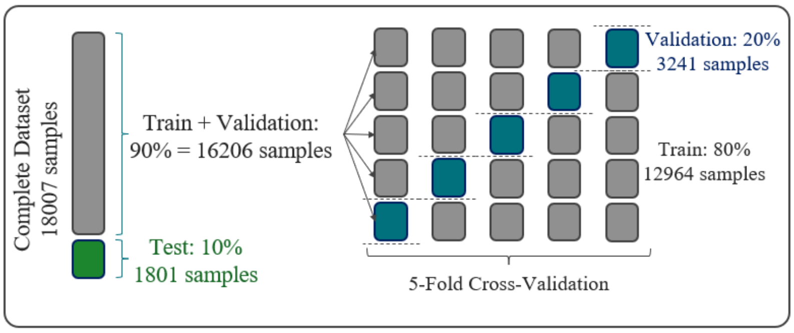


Figure 2. Dataset split for 5-fold cross-validation.

layers are required. Seeking a good compromise so that the network can learn complex representations, the number of hidden layers in the MLP was set to three. Still, the number of neurons in each hidden layer has a significant influence in MLP performance. Therefore, the NAS techniques described in the following section were only applied to these numbers, as optimizing fewer hyperparameters with greater impact makes the optimization process more efficient.

3. Fundamentals of Neural Architecture Search

Finding the optimal hyperparameters for an ML model has always been an important, yet demanding and time-consuming task. Recently, new methods have been developed to automate these tasks, greatly reducing the human effort required to optimize models [Feurer and Hutter 2019] and thus creating the field of Automated Machine Learning (AutoML). The positive results obtained by NNs and DL in recent years motivated the development of Neural Architecture Search (NAS), a subfield of AutoML.

NAS methods can be defined by three dimensions [Elsken et al. 2019]: a *search space* $\mathcal{S}_S$, which bounds the possible architectures evaluated during NAS; a *performance estimation strategy*, which defines an *objective function* C used to evaluate model performance during the search process; and a *search strategy*, which determines how the algorithm explores the search space. The search space and performance estimation strategy are manually defined to obtain the best optimization performance for the shortest execution times and to ensure that the models generated by this process are able to generalize to unseen data.

In this paper, the search strategy is defined by iterative algorithms that follow the same general structure. These methods depend on a *search history* $\mathcal{H}$, an initially empty data structure that stores the models architectures $\mathcal{M}$ and their performance, obtained by evaluating $C(\mathcal{M})$, which is to be minimized (or maximized) during optimization. The search process consists of a predetermined number k of iterations, also called *trials*, whose high-level overview is presented in the Algorithm 1. There are various strategies that can be employed in NAS. In this paper, three search strategies are evaluated: Random Search, Simulated Annealing, and Bayesian Optimization.

Random Search. Random Search (RS) is implemented by sampling architectures randomly from $\mathcal{S}_S$, aiming to find the architecture which most closely reaches the goal of the optimization process, as described in Algorithm 2. RS is most commonly used as baseline for comparison with other methods, yet has been shown to achieve performance similar to that of state-of-the-art NAS algorithms on some specific problems [Sciuto et al. 2019].

Algorithm 1: Trial

```
Choose $\mathcal{M} \in \mathcal{S}_S$ according to the search strategy;
Train $\mathcal{M}$;
Evaluate $C(\mathcal{M})$;
Append $\{\mathcal{M}, C(\mathcal{M})\}$ to $\mathcal{H}$;
```

Algorithm 2: Random Search

```
Input: search space $\mathcal{S}_S$, objective function $C$
Output: search history $\mathcal{H}$
$\mathcal{H} \leftarrow \emptyset$;
for $i = 1, 2, \ldots, k$ do
    $\mathcal{M} \leftarrow$ random($\mathcal{S}_S$);
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{\{\mathcal{M}, C(\mathcal{M})\}\}$;
end
```

Simulated Annealing. Simulated Annealing (SA) is a local optimization algorithm analogous to the metallurgical process annealing. SA relies on the definition of a *neighborhood* $n(\mathcal{M})$ of a model architecture $\mathcal{M}$, which must be defined during the algorithm implementation so that neighbor models are deemed to possess similar architectures. In each trial, a random neighbor architecture $\mathcal{M}' \in n(\mathcal{M})$ is chosen, evaluated and compared to the current architecture $\mathcal{M}$. If $\mathcal{M}'$ is better than $\mathcal{M}$ (i.e. $C(\mathcal{M}') < C(\mathcal{M})$), it is chosen as the next architecture; if it is not (i.e. if $C(\mathcal{M}') \geq C(\mathcal{M})$), its acceptance probability is given by

$$p(\mathcal{M}', \mathcal{M}, T) = \exp\left(-\frac{C(\mathcal{M}') - C(\mathcal{M})}{T}\right).$$

The *temperature* T is a parameter that regulates the probability of accepting worst models and decreases after each trial, commonly by exponential decay, in which the current temperature is multiplied by a constant decay rate α, $0 < \alpha < 1$. Both α and the initial temperature T_0 must be tuned to each specific problem. The general procedure is shown in Algorithm 3.

Due to the temperature decay mechanism, SA can explore less promising model architectures in early trials, yet still converge to a local minimum, since the probability of accepting a worse model decreases with the temperature. It is also relatively simple to implement and use, as the only parameters that need to be tuned are T_0 and α. However, as a local search algorithm, it can only consider architectures similar to the current one, according to the definition of its neighborhood n. Thus, it has limited capacity to explore the search space in a limited number of trials and is heavily dependent on $\mathcal{M}_0$. A more thorough discussion on Simulated Annealing can be found in [van Laarhoven and Aarts 1987].

Bayesian Optimization. Bayesian Optimization (BO) consists of two key components: a probabilistic surrogate model S of the objective function C; and a policy P, denoted as *acquisition function*, for selecting new parameters based on the surrogate model. In each trial, an evaluation of C updates the surrogate model S, allowing P to select a new architecture $\mathcal{M}$ most likely to achieve the objective of the optimization for the next trial. The general procedure is shown in Algorithm 4.

Algorithm 3: Simulated Annealing

```
Input: search space $\mathcal{S}_S$, objective function $C$, neighborhood $n$, initial
       architecture $\mathcal{M}_0$ and temperature $T_0$, temperature decay rate $\alpha$
Output: search history $\mathcal{H}$
$\mathcal{M} \leftarrow \mathcal{M}_0$;
$T \leftarrow T_0$;
$\mathcal{H} \leftarrow \{\{\mathcal{M}_0, C(\mathcal{M}_0)\}\}$;
for $i = 1, 2, \ldots, k-1$ do
    $\mathcal{M}' \leftarrow \texttt{random}(\{\mathcal{M}' \in n(\mathcal{M})\})$;
    if $(C(\mathcal{M}') < C(\mathcal{M}))$ or $(p(\mathcal{M}', \mathcal{M}, T) > \texttt{random}([0,1]))$ then
        $\mathcal{M} \leftarrow \mathcal{M}'$;
    end
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{\{\mathcal{M}, C(\mathcal{M})\}\}$;
    $T \leftarrow \alpha \cdot T$;
end
```

Algorithm 4: Bayesian Optimization

```
Input: search space $\mathcal{S}_S$, objective function $C$, initial architecture $\mathcal{M}_0$,
       surrogate model $S$, acquisition function $P$
Output: search history $\mathcal{H}$
$\mathcal{M} \leftarrow \mathcal{M}_0$;
$\mathcal{H} \leftarrow \{\{\mathcal{M}_0, C(\mathcal{M}_0)\}\}$;
Initialize the surrogate model $S$;
for $i = 1, 2, \ldots, k-1$ do
    $\mathcal{M} \leftarrow \arg\max_{\mathcal{M}' \in \mathcal{S}_S} P(\mathcal{M}', S)$;
    $\mathcal{H} \leftarrow \mathcal{H} \cup \{\{\mathcal{M}, C(\mathcal{M})\}\{$;
    $S \leftarrow$ updateSurrogate($S$, $\{\mathcal{M}, C(\mathcal{M})\}$);
end
```

Despite being more complex, BO presents two major advantages over SA. Firstly, it can evaluate any configuration in the search space in any trial, thus not being limited to evaluating architectures in the neighborhood of the current one. Secondly, it reduces the number of evaluations of the objective function (often expensive to evaluate) to achieve the optimization objective due to the use of the surrogate model.

The surrogate model is used as an estimate of the objective function C, and can be generated in a number of ways. Often Gaussian Process regression is used [Frazier 2018]. However, there has been a rise in the use of Tree-structured Parzen Estimator (TPE), as it is more flexible to non-numerical search spaces, is able to scale to bigger search spaces with a smaller computational cost and has been shown to achieve better performance on some optimization problems [Feurer and Hutter 2019, Bergstra et al. 2011]. Various choices are also available for the acquisition function, the most common being Expected Improvement (EI). This method is based on selecting the model with the best estimated performance at each turn. EI is calculated using the surrogate model, and the architecture with the largest EI is selected for evaluation in the next trial. A more thorough discussion on Bayesian Optimization can be found in [Frazier 2018, Bergstra et al. 2011].

4. Experiments and Results

The algorithmic performance evaluation was conducted in two stages: (i) an initial comparison of all three aforementioned techniques using sensible parameter configurations; and (ii) further analysis of the impact of different parameters for the best performing algorithm in the previous stage. In both stages, it is necessary to first define a common search space and performance estimation strategy.

Search Space Definition. This paper focuses exclusively on MLPs with three hidden layers (HLs), aiming to determine the optimal number of neurons in each layer. The number $\mathcal{N}_i$ of neurons in the i-th hidden layer ranges from 20 to 4470 with a step of 50, i.e. $\mathcal{N}_i \in N = \{20, 70, \ldots, 4470\}$, so reasonable sized models can be obtained in a space of search neither too small (which would constrain the algorithms) nor too large (which would hamper optimization). The search space is thus $\mathcal{S}_S \in \mathbb{N}^3 = N \times N \times N$.

Performance Estimation Strategy. The performance of the models generated by each optimization algorithm was evaluated to assess their relative effectiveness. A set of standard model hyperparameters, described in Table 1, remained fixed throughout the initial experiments, with the only variable parameter being the number of neurons in each HL of the MLP assigned by the optimization algorithm. During optimization, all models were compared based on the average mean squared error (MSE) across the five cross-validation folds, which was used as the objective function C. Cross-validation was implemented during optimization so the optimization algorithms do not develop a bias towards the validation set, and thus propose models able to generalize. After each optimization, model weights were reset and the best architecture in the search history $\mathcal{H}$ was trained on all five cross-validation folds (the train/validation set) for 5000 epochs and evaluated on the holdout test set. This allows for direct model performance comparison despite different parameter configurations and techniques employed during optimization.

Table 1. MLP hyperparameters for initial experiments.

Number of HLs	Training Epochs	Batch Size	Optimizer	Learning Rate	Activation Function
3	200	800	Adam	0.001	ReLU

4.1. Comparison Between the Algorithms

For all SA, BO, and RS experiments, the number of trials executed by each algorithm was kept constant at 500, i.e $k = 500$. Given that model training and evaluation are the most demanding components of the optimization process, this ensures all algorithms have a fixed computational budget and ensures a fair comparison of their optimization efficiency. In this paper, SA employed an exponentially decaying temperature profile, which depends on a starting temperature T_0 and a temperature decay rate α. For the initial experiment, these parameters were set to $T_0 = 0.1$ and $\alpha = 0.995$. These values were manually tuned through earlier experiments to ensure a balanced compromise between exploration in the initial trials (when temperature is high) and exploitation in the final ones (when temperature is low). The neighborhood function $n(\mathcal{M})$ is defined as a random choice within the set of architectures $\mathcal{M}_i, 1 \leq i \leq 26$, which differ from $\mathcal{M}$ by ± 50 in any combination of hidden layers, so that the Manhattan distance from $\mathcal{M}$ to $\mathcal{M}_i$ is either 50, 100 or 150. BO has a number of different parameters, such as the choice of surrogate probabilistic model, acquisition function and various parameters associated with them. The chosen surrogate model was TPE, and the acquisition function, EI, for which the standard configuration supplied in the Optuna framework [Akiba et al. 2019] was employed.

Under these conditions, SA, BO, and RS were executed and evaluated by analyzing NN performance in the validation set. Figure 3 gives an overview of the model architectures explored by each algorithm. Overall, the number of parameters in the analyzed architectures maintained the same order of magnitude across all processes, with three notable results arising. Firstly, the architectures investigated by SA had a significantly higher validation error than those by BO or RS; indeed, Figure 3b shows that even the best performing model obtained by SA had an MSE above the interquartile range of RS. Secondly, Figure 3a shows SA only analyzed architectures with a very close number of parameters to each other due to its local search strategy, highlighting its dependency on the initial architecture in order to succeed. Thirdly, BO shows a clear pattern of reducing both the number of parameters and the validation error in later trials, indicating the process converges to more promising regions of the search space; both Figure 3a and 3b highlight this, indicating that BO not only had on average models with lower error but also had the model with lowest error.

Figure 4 illustrates the evolution of the minimum MSE obtained by the search methods during optimization, in which the minimum MSE obtained up to a given trial is shown for each algorithm. Although SA optimized the error at a decent rate, its heavy reliance on the initial architecture resulted in significantly worse results than RS, given the global nature of this optimization problem. It also shows BO outperforming RS by decreasing validation error faster and by achieving the lowest MSE, as noted previously.

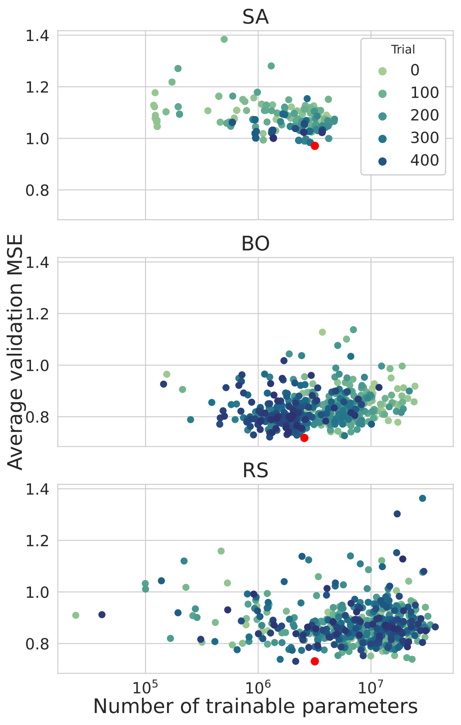
Table 2 reports the best architecture found with each algorithm and its MSE obtained by cross-validation during optimization and by evaluating on the holdout test set. These results show a large difference between the validation and test MSEs. However, BO still obtained the lowest test MSE.

Table 2. Best models found with SA, BO and RS.

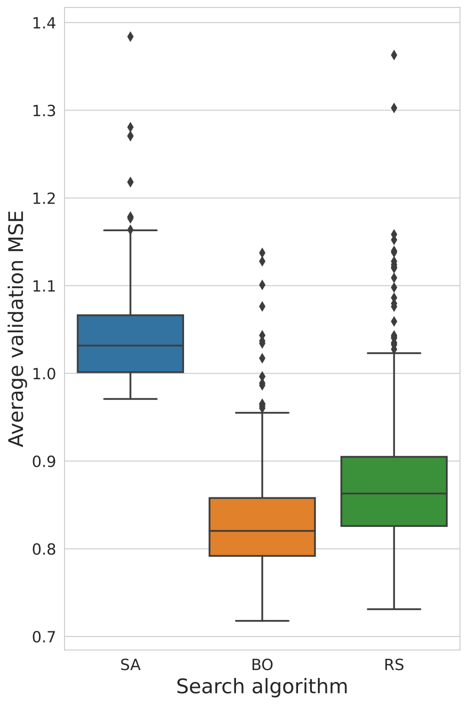
Optimization algorithm	**MLP architecture**			**Validation MSE**	**Test MSE**
	HL 1	HL 2	HL 3		
SA	2520	470	4150	0.970828	1.4783065
BO	370	2870	520	0.717813	1.4161978
RS	70	4370	420	0.731122	1.4194246

4.2. Further Testing of Bayesian Optimization

According to the comparative experiment, BO performed best. Therefore, it was chosen for further inspection to assess the influence of the number of training trials and epochs over the optimization process while also evaluating the test set performance for the obtained models.



(a) Model performance across the optimization, highlighting the best performer in red.



(b) Distribution of average cross validation MSE.

Figure 3. Statistics of models analyzed during optimization.

The initial experiment analyzed the behaviour of BO with more trials, and assessed the correlation between the validation error during optimization and the test error. Figure 5 illustrates a longer run of Bayesian Optimization, with 1200 trials and the same parameters presented in Table 1. Every time a model with better MSE was found, its architecture was saved and trained for 5000 epochs, then evaluated on the test set, and its test error was plotted. It is evident that the validation MSE – the optimization objective function – and test MSE are incompatible, since even close to the last trial, a decrease in the validation error caused a significant increase in the test error.

The incompatibility between validation and test MSE was hypothesized to stem from a hyperparameter configuration that did not allow validation error to serve as an appropriate estimator of test performance. In particular, the disparity between number of training epochs (200 during optimization, against 5000 during testing) was believed to play a significant role. Therefore, another experiment was conducted, maintaining all parameters from Table 1 constant except for the number of training epochs, which took up values of 100, 200, 300, 400 and 500. The number of trials was fixed at 500.

Figure 6 shows test MSE of the models obtained in this experiment, which highlights two distinct patterns. On the one hand, with up to 300 epochs during optimization, a strong negative correlation exists between the number of epochs and test error; however, the same relationship exists between the number of trainable parameters and test error,

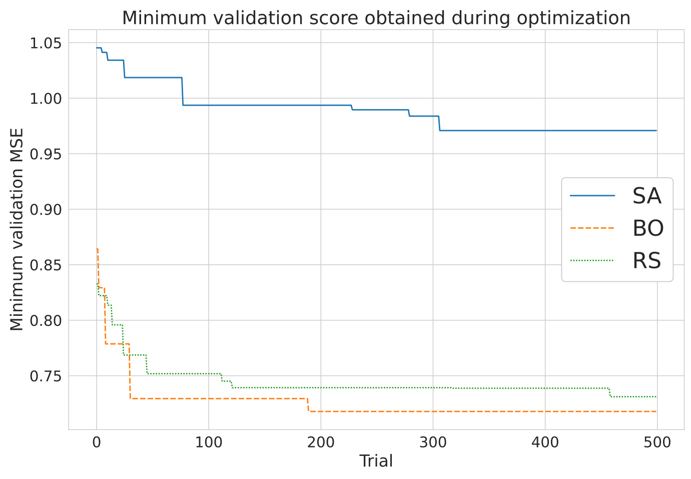


Figure 4. Minimum validation MSE obtained during the optimization process for SA, BO and RS.

indicating the analysis of bigger, more complex models during optimizations with greater number of epochs was responsible for the reduction in test loss. On the other hand, for 400 and 500 epochs, the previous pattern is disrupted: the result of optimization with 400 epochs performed worse than the one with 300 epochs despite having more parameters; and the result with 500 epochs was the best performer of this experiment in the validation and test sets while also possessing the smallest number of parameters. Thus, the increase in the number of training epochs during optimization had a positive effect on the optimization process overall, but results indicate erratic behaviour which suggests that the performance estimation strategy is still not completely adequate.

Overall, the best performing model – with 220 neurons in the first HL, 3120 in the second and 1520 in the third – was obtained in the experiment with 1200 trials, on trial 830, and achieved a test MSE of 1.3412.

5. Conclusion

This study aimed to obtain an optimal MLP model for offshore platform offset prediction employing NAS. Three methods were considered: Simulated Annealing, Random Search and Bayesian Optimization. SA had the worst performance, followed by RS and BO, which yielded the best result.

Despite the good results obtained by BO, a large difference between the validation error during optimization and the final test error motivated additional experiments to examine the influence of the number of trials and epochs in the optimization. Increasing the number of trials provided diminishing returns after the first 500 trials; nevertheless, the overall best model was found on trial 830. Moreover, increasing the number of training epochs during optimization to a value closer to that used during evaluation provided a significant improvement to the test error, highlighting the trade-off between computational cost and search performance.

Ultimately, despite reaching satisfactory results, a clear mismatch between the MSEs of validation during optimization and test warrants further research on the influence

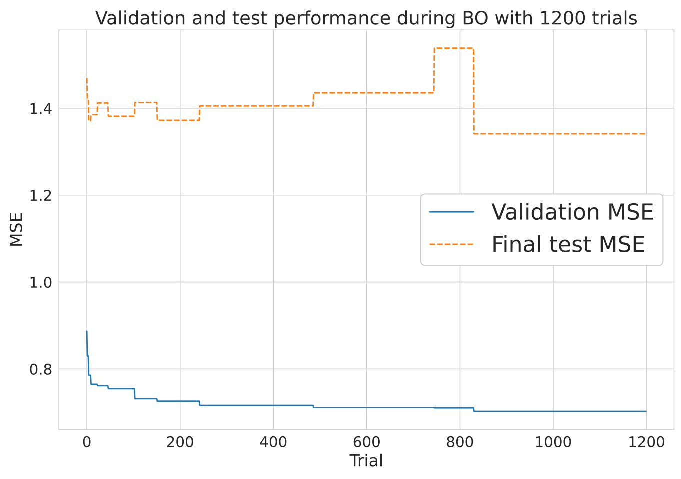


Figure 5. Validation MSE for the best model obtained during optimization compared to the training test MSE obtained by the same model.

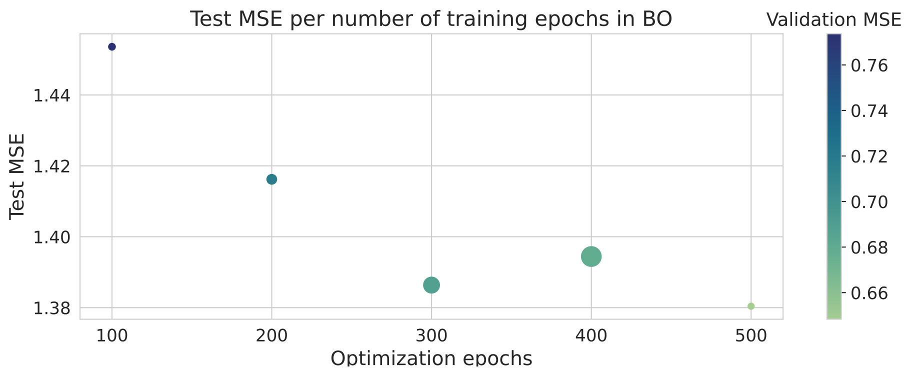


Figure 6. Test MSE for best model in BO runs with differing numbers of training epochs. Point size represents the number of parameters in the model.

of the performance estimation strategy over the optimization algorithms. Additionally, a more detailed comparison of the analyzed algorithms, subject to a greater variety of problems, would bring deeper insights into the advantages and disadvantages of each search strategy and of their generalization potential.

Acknowledgments

This work was financed in part by the *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES Finance Code 001), Brazil, ANP/PETROBRAS, Brazil (project N. 21721-6) and PTDF, Nigeria (Award N. PTDF/ED/PHD/SAM/977/19). We also gratefully acknowledge partial support from CNPq (research grants 310085/2020-9 and 310127/2020-3).

References

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. *CoRR*, abs/1907.10902.

Bergstra, J., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 1–9.

Cotrim, L. P., Oliveira, H. B., Filho, A., Santos, I., Barreira, A., Tannuri, E. A., Costa, A. H. R., and Gomi, E. S. (2021). Neural Network Meta-Models for FPSO Motion Prediction from Environmental Data. In *Proc. of the Int. Conf. on Offshore Mechanics and Arctic Engineering*.

Elsken, T., Metzen, J. H., and Hutter, F. (2019). Neural architecture search: A survey. *Journal of Machine Learning Research*, 20(55):1–21.

Feurer, M. and Hutter, F. (2019). *Hyperparameter Optimization*, pages 3–33. Springer International Publishing.

Frazier, P. I. (2018). A tutorial on bayesian optimization. arXiv:1807.02811.

Goffe, W. L., Ferrier, G. D., and Rogers, J. (1994). Global optimization of statistical functions with simulated annealing. *Journal of Econometrics*, 60(1):65–99.

Gumley, J. M., Henry, M. J., and Potts, A. E. (2016). A novel method for predicting the motion of moored floating bodies. In *35th Int. Conf. on Ocean, Offshore and Arctic Engineering*, OMAE2016-54674. DOI = 10.1115/OMAE2016-54674.

Jin, H., Song, Q., and Hu, X. (2018). Efficient neural architecture search with network morphism. *CoRR*, abs/1806.10282.

Liu, H., Simonyan, K., and Yang, Y. (2018). DARTS: differentiable architecture search. *CoRR*, abs/1806.09055.

Märtens, M. and Izzo, D. (2019). Neural network architecture search with differentiable cartesian genetic programming for regression. *CoRR*, abs/1907.01939.

Mazaheri, S., Mesbahi, E., Downie, M., and Incecik, A. (2003). Seakeeping analysis of a turret-moored FPSO by using artificial neural networks. In *Proc. of the Int. Conf. on Offshore Mechanics and Arctic Engineering*. DOI = 10.1115/OMAE2003-37148.

Nishimoto, K., Fucatu, C. H., and Masetti, I. Q. (2002). Dynasim—A Time Domain Simulator of Anchored FPSO. *J. Offshore Mech. Arct. Eng.*, 124(4):203–211.

Pinos, M., Mrazek, V., and Sekanina, L. (2021). Evolutionary neural architecture search supporting approximate multipliers. *CoRR*, abs/2101.11883.

Sciuto, C., Yu, K., Jaggi, M., Musat, C., and Salzmann, M. (2019). Evaluating the search phase of neural architecture search. *CoRR*, abs/1902.08142.

van Laarhoven, P. J. M. and Aarts, E. H. L. (1987). *Simulated Annealing: Theory and Applications*. Springer Netherlands.

White, C., Neiswanger, W., and Savani, Y. (2019). Bananas: Bayesian optimization with neural architectures for neural architecture search. arXiv:1910.11858.

Zoph, B. and Le, Q. V. (2016). Neural architecture search with reinforcement learning. arXiv:1611.01578.

Zoph, B., Vasudevan, V., Shlens, J., and Le, Q. V. (2017). Learning transferable architectures for scalable image recognition. *CoRR*, abs/1707.07012.