Check for updates

An Automated Machine Learning-Genetic Algorithm Framework With Active Learning for Design Optimization

Opeoluwa Owoyele[1]
Energy Systems Division,
Argonne National Laboratory,
9700 S. Cass Avenue,
Lemont, IL 60439
e-mail: oowoyele@anl.gov

Pinaki Pal
Energy Systems Division,
Argonne National Laboratory,
9700 S. Cass Avenue,
Lemont, IL 60439
e-mail: pal@anl.gov

Alvaro Vidal Torreira
Parallel Works Inc.,
222 Merchandise Mart Plz., Suite 1212,
Chicago, IL 60654
e-mail: alvaro@parallelworks.com

The use of machine learning (ML)-based surrogate models is a promising technique to significantly accelerate simulation-driven design optimization of internal combustion (IC) engines, due to the high computational cost of running computational fluid dynamics (CFD) simulations. However, training the ML models requires hyperparameter selection, which is often done using trial-and-error and domain expertise. Another challenge is that the data required to train these models are often unknown a priori. In this work, we present an automated hyperparameter selection technique coupled with an active learning approach to address these challenges. The technique presented in this study involves the use of a Bayesian approach to optimize the hyperparameters of the base learners that make up a super learner model. In addition to performing hyperparameter optimization (HPO), an active learning approach is employed, where the process of data generation using simulations, ML training, and surrogate optimization is performed repeatedly to refine the solution in the vicinity of the predicted optimum. The proposed approach is applied to the optimization of a compression ignition engine with control parameters relating to fuel injection, in-cylinder flow, and thermodynamic conditions. It is demonstrated that by automatically selecting the best values of the hyperparameters, a 1.6% improvement in merit value is obtained, compared to an improvement of 1.0% with default hyperparameters. Overall, the framework introduced in this study reduces the need for technical expertise in training ML models for optimization while also reducing the number of simulations needed for performing surrogate-based design optimization. [DOI: 10.1115/1.4050489]

Keywords: design optimization, internal combustion engine, machine learning, genetic algorithm, super learner, active learning, hyperparameter tuning

1 Introduction

In the development of novel design strategies and next-generation combustion technologies that boost fuel economy and comply with stringent emission regulations, one key component will be the optimization of engines running on such novel technologies. An emerging trend in this area is to use computational models as experimental surrogates to obtain promising engine designs that are subsequently verified against experiments. Using simulation-driven design optimization (SDDO) has the potential to minimize costs while allowing for the exploration of a broad design space, including the optimization of design parameters that are hard to vary in experiments. Nonetheless, more research into making design optimizers efficient is needed, due to unsolved problems that hinder SDDO's use in IC engine-type applications. One challenge is that in general, optimization of IC engines involves optimizing a primary objective function (often the specific fuel consumption) while satisfying soft constraints imposed by secondary objectives (typically relating to emissions and engine mechanical limits). In many cases, the design parameters that lead to the best primary objective excessively breach the secondary constraints. Violating these constraints can lead to unwanted phenomena, such as knock and high levels of pollutants. A more critical problem, however, is that SDDO involves running several computational fluid dynamics (CFD) simulations, which are computationally expensive. Therefore, traditional optimizations that run iteratively over generations can take several months to obtain optimum design parameters.

Genetic algorithm (GA)-based optimization [1–7], design of experiments (DoE) [8–11], and swarm-based optimization [12–14] have been employed for IC engine design in previous studies. DoE uses an appropriate space-filling technique to explore the design surface by running several simulations. Without the addition of high-order terms and cross-terms, DoE methods are only capable of capturing linear relationships, often leading to a loss of accuracy. GA involves running a small batch of CFD simulations over successive generations. Here, designs that produce poor fitness values are eliminated, while the traits of designs that yield superior fitness are encouraged to propagate through generations. GA, while usually leading to better optimum designs compared to DoE, often takes a long time to converge. Swarm-based approaches such as particle swarm optimization (PSO) [15] and artificial bee colony [16], based on the concept of swarm intelligence, involve a large number of swarm agents collaboratively seeking to find the optimum on a design surface. A major disadvantage of these optimizers for IC engine applications is that they are designed to operate with fairly large swarms and sometimes do not sufficiently explore the design space when smaller swarms are used. For engine applications where only a few simulations are typically run at the same time due to computational costs, such optimizers often converge prematurely to a local optimum.

Some previous studies have utilized machine learning (ML) techniques to improve the performance of the optimizers discussed above. Kavuri and Kokjohn [17] introduced an approach for speeding up GA optimization using Gaussian process regression (GPR). The GPR model was used as a surrogate for CFD simulations in regions of the design space where the uncertainty in the prediction of the regression model was less than a pre-specified tolerance.

[1]Corresponding author.

Contributed by the Internal Combustion Engine Division of ASME for publication in the Journal of Energy Resources Technology. Manuscript received February 23, 2021; final manuscript received February 26, 2021; published online April 9, 2021. Editor: Hameed Metghalchi.

Downloaded from http://asmedigitalcollection.asme.org/energyresources/article-pdf/143/8/082305/6680013/jert_143_8_082305.pdf by Purdue University at West Lafayette user on 17 April 2021

Their study showed a reduction in clock-time at the expense of a small reduction in fitness value. Bertram and Kong [12] explored the use of support vector machines to enable a particle swarm optimizer escape local minima and thus accelerate its convergence to the global optimum. By updating the position of the best particle in situations where the regression model found a better optimum, their method was able to achieve better convergence compared to a standalone particle swarm optimizer. Owoyele and Pal [18,19] introduced a novel sequential optimizer, ActivO, which works as an adaptive surrogate technique. Here, a low-variance learner was used to explore the design surface while a high-variance learner simultaneously attempts to exploit promising regions of the design space. Their study showed that ActivO reduced the number of simulations needed to reach the design optimum could be reduced by as much as 80%.

As an alternative to these approaches, some studies have explored the use of static surrogate models for engine design optimization [20–23]. In these studies, a large number of simulations are run over the entire design space. Afterwards, an ML model is trained to predict the fitness as a function of the design parameters, and an optimizer (e.g., GA, PSO) is used to find the design optimum as predicted by the ML model. Moiz et al. [20] developed a workflow for engine design optimization, ML-GA, where an ensemble ML model called a super learner [24] was used as a surrogate for CFD simulations. A GA optimizer was then used to find the optimum based on the super learner-predicted fitness value. Since the simulations could be run in parallel, this study showed that the turnaround time could be reduced by as much as 95% compared to the conventional approach of coupling GA directly with CFD. Badra et al. [22] extended the ML-GA approach of Moiz et al. [20] to incorporate robustness of the optimum obtained using a grid gradient ascent (GGA) algorithm in place of GA. Also, an active learning technique for refining the solution in the neighborhood of the best fitness was performed. Here, 256 CFD simulations were run to generate a training dataset. After training a surrogate model and determining the global optimum using GGA, CFD simulations were run for the best five points, and these points were added to the training data. Training of the ML model and generation of five new optima was then repeated. This loop was continued until no further improvement in merit was obtained.

The present work builds on the work of Moiz et al. [20] and Badra et al. [22] and addresses some of the unresolved shortcomings of the ML-GA approach. A lingering problem lies in the selection of hyperparameters. For example, when training artificial neural networks (ANNs), untrained parameters that influence the performance of the neural network (such as the number of hidden layers, number of neurons per layer, and the learning rate) need to be chosen. For tree-based algorithms, the selection of parameters such as the number of trees and the maximum depth of the trees is required. These parameters are problem- and data-dependent, often requiring trial-and-error and/or a priori expert knowledge of the problem. Therefore, to enable widespread adoption of surrogate-based approaches in the design of next-generation engines, techniques that help democratize the use of such ML models need to be developed.

In this paper, a new automated ML-GA framework is introduced (AutoML-GA). In this approach, the untrained parameters (referred to as hyperparameters) of the ML models are optimized using a Bayesian optimization technique. Similar to the ML-GA technique, a super learner model is developed using optimized base learners. The super learner acts as a surrogate for the ground-truth function to be optimized. The results obtained in this paper are compared to those obtained using default hyperparameters from a common PYTHON-based ML library, Scikit-learn [25]. It is shown that the obtained training errors and fitness values are significantly superior to those obtained using the default hyperparameters. Besides, we extend the work of Badra et al. [22], showing that an active learning approach combined with hyperparameter optimization (HPO) can find the design optimum even when a reduced number of baseline simulations is used for the initial ML training. The potential impact of this paper is to enable engineers and researchers to seamlessly integrate ML-based optimization tools with existing workflows, while eliminating the inefficient ad hoc tweaking and heuristic tuning of model hyperparameters.

2 Automated Machine Learning-Genetic Algorithm

2.1 Machine Learning-Genetic Algorithm Details. Optimization of IC engines typically utilizes a GA, where the fitness values of a small number of designs are evaluated over successive generations using CFD simulations. Since CFD simulations are computationally expensive, this sequential process can take a long time to complete. ML-GA seeks to circumvent this problem by avoiding running simulations sequentially. The ML-GA approach starts by generating a large number of sample points within the pre-defined parameter space using Monte-Carlo sampling and running 3D CFD simulations for these generated samples. Using data from the simulations, a super learner is used to learn the fitness values as a function of the design parameters. With sufficiently low prediction errors, it is assumed that the super learner is a reliable surrogate for 3D CFD simulations. In that case, a global optimizer can be used to find the optimum based on the super learner predictions, as opposed to making use of time-consuming CFD simulations.

While traditionally, ML is carried out using a single learner such as an ANN or a support vector regressor, a super learner combines such learners into one single learner. The individual learners that make up the super learner are called the base learners. In previous studies employing ML-GA in engine optimization [15–17], the super learner was based on a software available as an add-on package to the statistical software R. In this work, the super learner is based on an in-house software built using PYTHON packages, such as SciPy [26], Scikit-learn [25], and NumPy [27]. The base learners used in this study are briefly described below.

Regularized polynomial regression (RPR): This learner uses polynomial transformations to obtain a new basis set from the original list of features. These features are then used as inputs to an L2-regularized linear regression model. The polynomial basis function transformation allows the learner to capture nonlinear behavior, while regularization helps reduce unrealistic oscillations at the extremities of the independent variable space.

Support vector regression (SVR) [28]: SVR performs regression by attempting to find the line that incurs the smallest cost. This is based on a margin defined by a tolerance value, ϵ. Points that are within this margin incur no cost, but points that violate it incur a penalty. Kernels are used to enable learning of nonlinear functions. A υ-SVR approach is used in this work. Here, the proportion of support vectors to data samples is chosen, and ϵ is adjusted accordingly.

Kernel ridge regression (KRR): KRR combines L2-regression with a kernel transformation to allow for nonlinear mappings. It shares similarities with SVR, but the loss function that is minimized in this case is the squared error.

Extreme gradient boosting (XGB) [29]: This learner is based on an ensemble of regression trees. Each regression tree contains a recursive bifurcation at nodes that splits the feature space into partitions and assigns constant values to these partitions. Gradient boosting is a method to iteratively combine weak regression trees to form a strong learner. XGB is an implementation of gradient boosting with advantages such as L1 and L2 regularization to improve generalization, the use of second-order gradients in computing loss function derivatives, and faster computation times.

Artificial Neural Networks: ANNs are loosely inspired by biological neural networks. They consist of successive layers parameterized by weights and biases with an activation function to allow them to learn nonlinear mappings.

The base learners described above were trained using the Scikit-learn [25] library in PYTHON. These base learners have hyperparameters, which are not learned during training but are required to

Downloaded from http://asmedigitalcollection.asme.org/energyresources/article-pdf/143/8/082305/6680013/jert_143_8_082305.pdf by Purdue University at West Lafayette user on 17 April 2021

Table 1 List of base learners and their default hyperparameters

RPR	
Regularization weight, α	1.0
Polynomial deg	2
SVR	
Parameter to control the number of support vectors, υ	0.5
Penalty parameter, C	1.0
Kernel	Radial basis function
Kernel coefficient, γ	Inverse of the number of features
KRR	
Regularization parameter, α	1.0
Kernel	Radial basis function
Gamma parameter, γ	None
XGBoost (XGB)	
Number of trees	1000
Learning rate	0.1
Maximum tree depth	6
ANN	
Number of layers	1
Number of neurons in each hidden layer	100
Activation function	relu
Solver	Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS)
L2 regularization parameter, α	0.0001
Tolerance to determine if training should be stopped early, tol	0.0001

be set by the user. The hyperparameters of the different base models are shown in Table 1. These parameters represent the default values in Scikit-learn, except KRR where the kernel was changed from "linear" to a "radial" basis function.

The super learner prediction is a weighted sum of the base learner predictions, given by

$$\varphi_{SL} = W_{RPR}\varphi_{RPR} + W_{SVR}\varphi_{SVR} + W_{KRR}\varphi_{KRR} + W_{XGB}\varphi_{XGB} + W_{ANN}\varphi_{ANN} \quad (1)$$

In the above equation, φ represents the prediction of an ML model, while W represents the weights of the learners. After the base learners have been trained, a non-negative least square solver is used to find the values of W that yield the best test errors. In this work, 150 data points were used to train the ML model. For such a small dataset, setting aside a portion as a test dataset would provide a highly inaccurate estimate of the test error and would be wasteful. Therefore, to obtain a reasonable estimate of the test error, a k-fold cross-validation (CV) strategy was adopted. The data were divided into ten equal folds. Ten training iterations were then performed, and for each iteration, a specific fold was chosen as the test set, while the remaining nine folds were used as the training data. After the k-fold CV process was completed, the test error across all the different folds was averaged to get an estimate of the test error from a given base learner. An illustration of this process is shown in Fig. 1.

2.2 Automated Hyperparameter Selection. When training ML models, it is not guaranteed that the default hyperparameters (for this case, shown in Table 1) are optimal for training. These hyperparameters often control how fast the learners are trained, the degree of regularization, and other factors that affect the bias-variance tradeoff of the learners. Different applications require different hyperparameters. Furthermore, even for the same problem, the hyperparameters can be different based on the subset of samples selected or the data size. This presents a daunting problem. For small datasets, the variance of the model needs to be reduced at the expense of bias, while in the abundance of data, a higher variance can be utilized to reduce test errors. In this work, in addition to the basic ML-GA approach, an automated workflow for finding the best hyperparameters was developed. After data generation and selection of base learners, an intermediate step in introduced, wherein the hyperparameters of the base learners are optimized. The objective of HPO is to find the combination of hyperparameters that produces the smallest k-fold CV error. In situations where there is only one hyperparameter to be optimized (e.g., Lasso regression), it is possible to use a simple line search to find the best hyperparameters. However, the cost of line or grid search scales exponentially with the number of hyperparameters. Another alternative involves performing a random search, but this approach relies on chance with no guarantee of success.

Bayesian optimization [30,31], which is used in this study, offers a robust and methodological way to perform HPO. The Bayesian

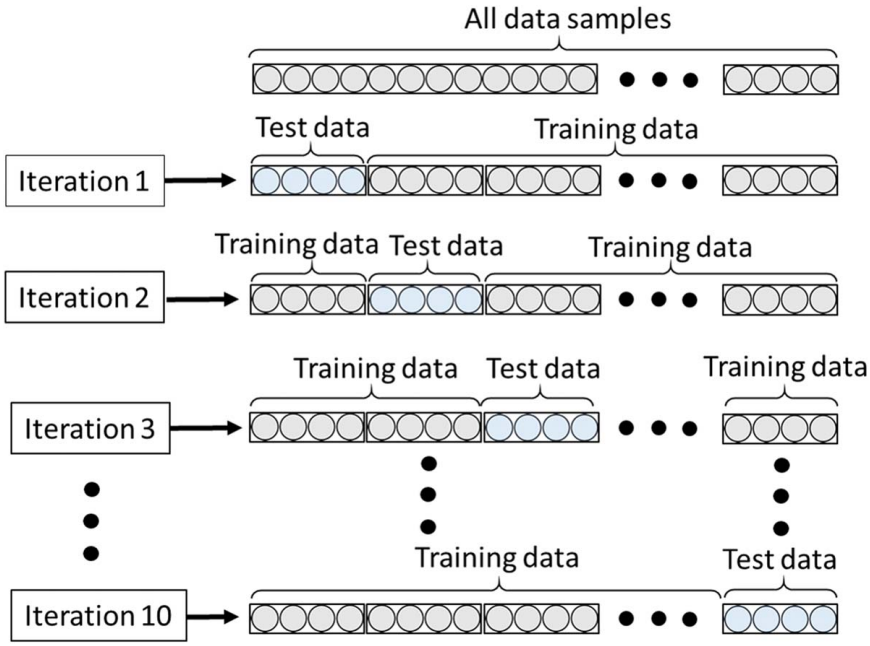

Fig. 1 An illustration of the k-fold CV approach used in this study

Downloaded from http://asmedigitalcollection.asme.org/energyresources/article-pdf/143/8/082305/6680013/jert_143_8_082305.pdf by Purdue University at West Lafayette user on 17 April 2021

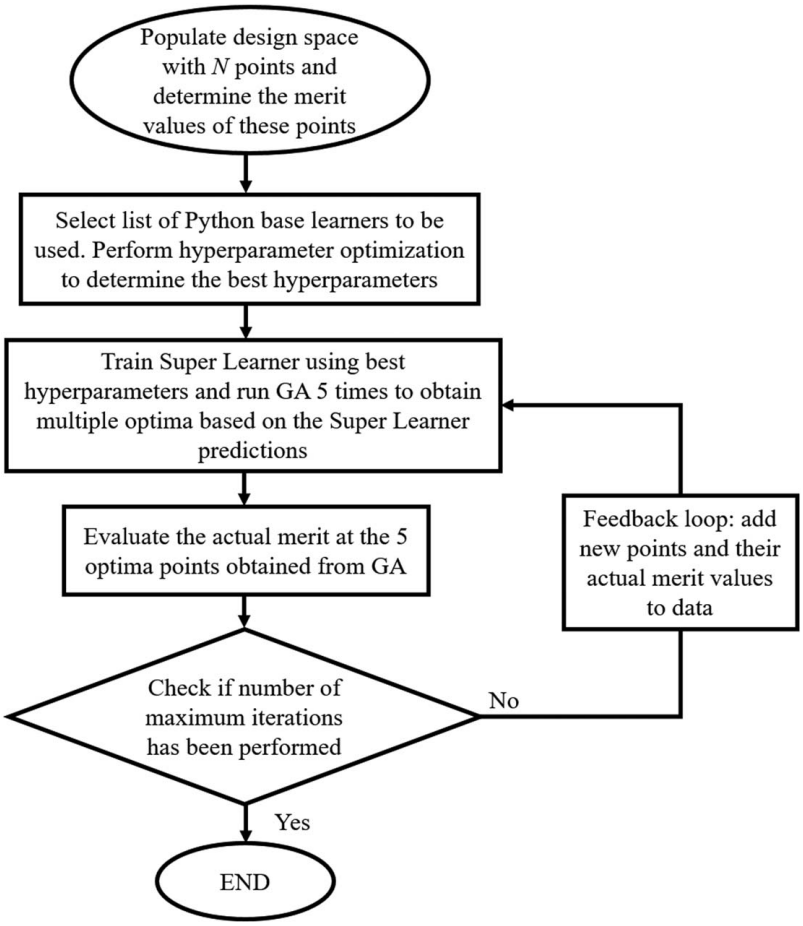

Fig. 2 Flowchart of the AutoML-GA workflow

optimizer used in this study was a PYTHON-based open source implementation [32]. Using Bayesian optimization, the hyperparameters shown in Table 1 are adjusted to reduce the CV errors of the base learners to their minimum values. When performing HPO, neither gradients nor a closed-form solution that relates the CV errors to the hyperparameters exists. Therefore, the objective function is treated as a black box and a surrogate, often GPR, is used to learn the CV error as a function of the hyperparameters. The model repeatedly evaluates the CV error using hyperparameters that are likely to yield improvement. The steps involved are outlined as follows:

(1) Start by randomly selecting N hyperparameters sets. Run the ML model using these hyperparameters and evaluate the CV error.
(2) Build a GPR model using all available ground-truth data. The trained GPR gives a prediction of the CV error as a function of the hyperparameters. Since GPR constructs a posterior distribution of functions, it also provides an estimate of the uncertainty in the predictions, usually in the form of a standard deviation.
(3) Construct the acquisition function, θ, given by

$$\theta = \mu + \kappa\sigma \tag{2}$$

In the above equation, μ is the mean merit value, while σ is the standard deviation as predicted by the GPR. While many different types of acquisition functions are available, the acquisition function shown above (known as upper confidence bound) is used in this work. κ is a parameter that controls the balance between exploration and exploitation.

(4) Use a global optimizer to find hyperparameters that maximize the acquisition function.
(5) Run the ML model using the set of hyperparameters from step 4 and obtain CV errors.
(6) Add new data, including hyperparameters and the obtained CV test error from step 5 to the dataset.
(7) Return to step 2. Repeat until convergence.

The acquisition function in step 3 incorporates two components. The first is the predicted CV error, while the second is the uncertainty in the predicted CV error. If the predicted CV error for a given set of hyperparameters is low, it may be desirable to exploit the area around such points to further exploit the region. On the other hand, regions that have high uncertainty (when little information is known about such regions) may also lead to better CV errors if they are explored. The acquisition function takes these two criteria into account by combining the predicted CV error and the uncertainty. The optimizer can be made to exploit more aggressively or explore more curiously by adjusting the value of κ. The value of κ used in this study was 2.576.

By performing HPO, the AutoML-GA approach introduced in this study can adapt to different applications and datasets in an automated fashion. Moreover, due to the small initial dataset size of 150 samples, an active learning approach was used to refine the solution in the vicinity of the global optimum, similar to Badra et al. [22]. GA was run five times on the trained super learner to obtain five points for evaluation in CFD. These points, once evaluated, are added to the training data. The optimization process is then repeated with this additional ground-truth information regarding the design space. The overall AutoML-GA approach is described using the schematic in Fig. 2.

Downloaded from http://asmedigitalcollection.asme.org/energyresources/article-pdf/143/8/082305/6680013/jert_143_8_082305.pdf by Purdue University at West Lafayette user on 17 April 2021

3 Problem Setup

Global optimizers often incorporate stochastic elements in their operation to allow for exploration. To make a meaningful projection of how well they perform in practice, multiple realizations or trials need to be performed to obtain averaged performance metrics. This presents a challenge in developing new optimizers for CFD applications, since these involve simulations that are time-consuming. As a result, carrying out several trials using CFD simulations becomes unrealistic. On the other hand, the results obtained from one or two CFD trials may be fortuitous and do not provide meaningful estimates of the robustness of an optimizer. Thus, while one potential way to test AutoML-GA would be to utilize 3D CFD simulations, this is avoided in the current study. In the present analysis, Moiz et al. [20] super learner surrogate model, which has been verified against CFD simulations, was assumed to provide ground-truth information and used as a surrogate for the CFD model. Thus, it is assumed that the super learner can be relied upon to provide similar fitness values as would be obtained if CFD simulations were run. This is a reasonable assumption since the super learner in the study by Moiz et al. [20] was built using a large dataset comprising 2000 engine CFD simulations covering the entire design space and was shown to have low CV test errors. This strategy allowed for multiple trials of AutoML-GA to be carried out to obtain reasonable estimates of its performance.

The 2048 3D CFD simulations of an IC engine from Ref. [15] were performed in CONVERGE [33]. The engine model was based on a turbocharged Cummins heavy-duty engine with exhaust gas recirculation (EGR) and a charged air cooler [34], operating at medium load conditions. While being a compression ignition engine, the engine operates on a gasoline-like fuel, for which a RON70 primary reference fuel blend surrogate was used in CFD simulations. A summary of the engine operating conditions is provided in Table 2, and complete details of the CFD model can be found in Refs. [15,29].

The optimization problem chosen to demonstrate the efficacy of AutoML-GA is the same as Ref. [15], i.e., to minimize the indicated specific fuel consumption (ISFC), while satisfying constraints on soot and NO_x emissions (SOOT and NO_x, respectively), peak cylinder pressure (PMAX), and the maximum pressure rise rate (MPRR). Nine input parameters relating to the in-cylinder flow field, thermodynamic conditions, and fuel injection are considered. These nine input parameters with their considered ranges are listed in Table 3.

In line with the previous study by Moiz et al. [20], the merit function is defined as follows:

$$\text{Merit} = 100 * \left\{ \frac{160}{\text{ISFC}} - 100 * f(\text{PMAX}) - 10 * f(\text{MPRR}) - f(\text{SOOT}) - f(\text{NO}_x) \right\} \tag{3}$$

Table 2 Brief summary of engine operating conditions [34]

Engine model	Cummins ISX15
Displacement	14.9 L
Bore	137 mm
Stroke	169 mm
Connecting rod	262 mm
Compression ratio	17.3:1
Engine speed	1375 rpm
Intake valve closing (IVC)	−137 °CA after top dead center (ATDC)
Exhaust valve opening	148 °CA ATDC
Injection duration	15.58 °CA
Mass of fuel injected	0.498 g/cycle
Fuel injection temperature	360 K
Global equivalence ratio	0.57

Table 3 Input parameters used for engine combustion optimization with their ranges [20]

Parameter	Description	Min	Max	Units
nNoz	Number of nozzle holes	8	10	–
TNA	Total nozzle area	1	1.3	–
Pinj	Injection pressure	1400	1800	bar
SOI	Start of injection timing	−11	−7	°CA ATDC
NozzleAngle	Nozzle half-inclusion angle	72.5	83.0	deg
EGR	EGR fraction	0.35	0.5	–
Tivc	IVC temperature	323	373	K
Pivc	IVC pressure	2.0	2.3	bar
SR	Swirl ratio	−2.4	−1	–

where

$$f(\text{PMAX}) = \begin{cases} \dfrac{\text{PMAX}}{220} - 1, & \text{if PMAX} > 220 \\ 0, & \text{if PMAX} \le 220 \end{cases}$$

$$f(\text{MPRR}) = \begin{cases} \dfrac{\text{MPRR}}{15} - 1, & \text{if MPRR} > 15 \\ 0, & \text{if MPRR} \le 15 \end{cases}$$

$$f(\text{SOOT}) = \begin{cases} \dfrac{\text{SOOT}}{0.0268} - 1, & \text{if SOOT} > 0.0268 \\ 0, & \text{if SOOT} \le 0.0268 \end{cases}$$

$$f(\text{NO}_x) = \begin{cases} \dfrac{\text{NO}_x}{1.34} - 1, & \text{if NO}_x > 1.34 \\ 0, & \text{if NO}_x \le 1.34 \end{cases}$$

Constraints of 220 g/kW h, 15 bar/deg, 0.0268 g/kW h, and 1.34 g/kW h are set on PMAX, MPRR, SOOT, and NO_x, respectively. The merit is designed to encourage the optimizer to find designs that do not exceed these constraints, by imposing penalties for constraint violations.

In this work, a total of 150 data points were randomly sampled within the design space to provide data for training the ML models. Using the data generated, a super learner model composed of the base learners shown in Table 1 was developed to predict each of the five intermediate outputs in Eq. (3) (SOOT, NO_x, MPRR, PMAX, and ISFC) as functions of the input parameters in Table 3. Based on the super learner model trained using these 150 data points, five optimum points were obtained. The ground-truth merit values at these optimum points were evaluated; the new data were added to the training dataset, and then the optimization process was repeated.

The process of ML training, finding new optima, ground-truth evaluation of optimum points, and inclusion of new points in the database (shown in the flowchart in Fig. 2) was continued for 20 iterations. In practice, it may be beneficial to terminate the loop in Fig. 2 once the difference between the ground-truth and the surrogate model at the design optimum falls below a pre-specified error. However, in this study, it was necessary to run all trials for the same number of iterations to make a meaningful comparison between different test cases. In total, three test cases were considered. In case A, the base learners in the super learner model were trained using the default hyperparameters in the Scikit-learn libraries listed in Table 1. For case B, HPO was performed (only once, based on the initial 150 data points) using Bayesian optimization to determine the best hyperparameters. In this case, unique optimal hyperparameters were determined for each base learner for the prediction of each of the individual outputs. Since there are five base learners (RPR, SVR, KRR, XGB, and ANN) and five intermediate outputs (ISFC, SOOT, NO_x, MPRR, and PMAX), a total of 25 groups of

Downloaded from http://asmedigitalcollection.asme.org/energyresources/article-pdf/143/8/082305/6680013/jert_143_8_082305.pdf by Purdue University at West Lafayette user on 17 April 2021

Table 4 Summary of three different optimization strategies considered in this study

Case	Description
Case A (ML-GA)	No HPO; default model hyperparameters are used
Case B (AutoML-GA)	HPO before the first iteration only
Case C (AutoML-GA)	HPO before the first iteration and after every five iterations subsequently

Table 5 Limits imposed on hyperparameters during Bayesian optimization

Base learner	Parameter	Min value	Max value	Log scale
RPR	Regularization weight, α	−6	0	Yes
RPR	Polynomial deg	1	10	No
SVR	Parameter to control the number of support vectors, υ	1×10^{-10}	1.0	No
SVR	Penalty parameter, C	−6	2.5	Yes
SVR	Kernel coefficient, γ	−6	0	Yes
KRR	Regularization parameter, α	−6	0	Yes
KRR	Gamma parameter, γ	−4	0	Yes
XGB	Number of trees	2	4	Yes
XGB	Learning rate	−3	0	Yes
XGB	Maximum tree depth	2	8	No
ANN	Number of neurons in each hidden layer	10	250	No
ANN	L2 regularization parameter, α	−6	0	Yes
ANN	Tolerance to determine if training should be stopped early, tol	−6	−2	Yes

optimal hyperparameters were obtained. Each group contained the best hyperparameters for predicting a specific output using a given base learner. Once the optimal hyperparameters were determined, they were used for successive iterations without any changes. Case C involved HPO like case B, but in addition to performing HPO before the first iteration, it was repeated after every five iterations. The differences between cases A, B, and C are summarized in Table 4, while the optimized hyperparameters with the respective ranges considered are shown in Table 5.

4 Results and Discussion

In this section, the performance of cases A, B, and C is presented and discussed. Since HPO leads to additional computational cost, it is important to factor this into the analysis to ensure that the savings AutoML-GA provides over ML-GA outweighs the cost. For cases B and C, each time HPO was performed; it was done using a time-limit of 4 h on 25 processors. The AutoML-GA feedback loop was allowed to run for 20 iterations in all cases. Therefore, case B incurs an additional cost of 4 h runtime due to HPO performed at the beginning before the first iteration. The cost of doing HPO for case C is 16 h of runtime since HPO is done at the beginning and every five iterations afterward. Case A incurs no additional runtime. As opposed to running one trial to test the optimization strategies, a total of ten trials were run for each case. Each trial corresponds to a different realization, where the optimization process is independently repeated. Due to the stochastic nature of some of the base learners (e.g., ANN, XGB) and the GA optimizer, each trial leads to different rates of convergence. In addition to randomness due to these, the data used for each trial were different. For each trial, a different dataset with a sample size of 150 was randomly drawn and used as the initial dataset for training during the first iteration. Thus, the different trials incorporate the different sources of randomness, which is expected during the practical deployment of optimizers. To make the comparison fair, each of the cases was initialized with the same 10 datasets over the 10 trials, so that the baseline merit was the same.

Figure 3 shows parity plots for cases A and C, comparing the actual values of ISFC against the predictions obtained using the base learners and the super learner developed in PYTHON. While different parity plots could be shown for training based on the dataset at any of the 20 iterations, the parity plot shown here is based on the initial dataset of 150 points. As indicated in Table 4, HPO is performed before the first iteration in case C. Therefore, these plots highlight the difference between case A and case C due to the

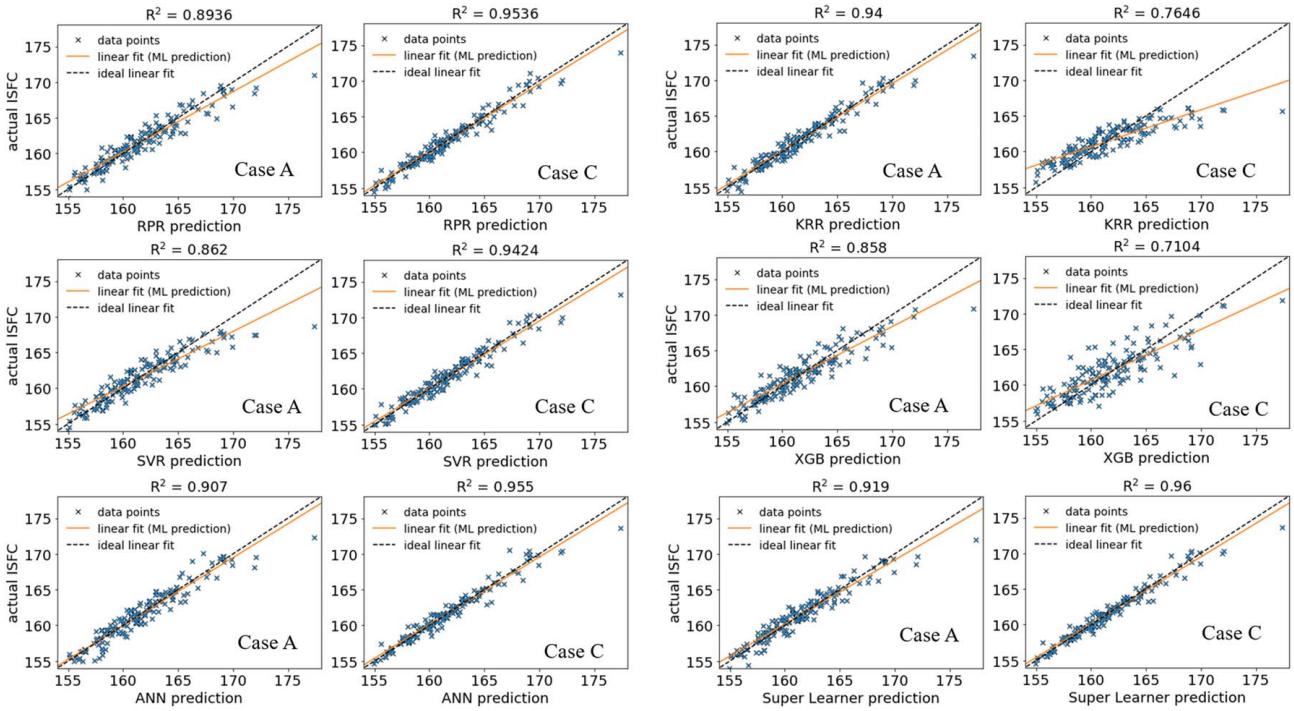

Fig. 3 ISFC parity plots for case A versus case C using various base learners and the super learner

Downloaded from http://asmedigitalcollection.asme.org/energyresources/article-pdf/143/8/082305/6680013/jert_143_8_082305.pdf by Purdue University at West Lafayette user on 17 April 2021

Table 6 The mean R^2 values over ten trials for case A and case C

	ISFC	SOOT	NO_x	MPRR	PMAX
ANN					
Case A	0.917	0.9165	0.975	0.953	0.9785
Case C	0.955	0.948	0.9893	0.967	0.9927
SVR					
Case A	0.864	0.8145	0.9297	0.9473	0.9795
Case C	0.9375	0.9287	0.989	0.9683	0.992
XGB					
Case A	0.6875	0.6787	0.869	0.71	0.7417
Case C	0.8223	0.7793	0.928	0.8604	0.899
KRR					
Case A	0.7793	0.6987	0.8867	0.907	0.958
Case C	0.9375	0.927	0.9893	0.968	0.9927
RPR					
Case A	0.8936	0.8525	0.9097	0.934	0.961
Case C	0.944	0.939	0.9834	0.967	0.993
Super learner					
Case A	0.9263	0.9214	0.977	0.959	0.983
Case C	0.9604	0.9585	0.9917	0.9736	0.9946

HPO done using the initial generated data before the first iteration was performed. Since case B and case C are identical after the first iteration (based on Table 4), results for case B are omitted. Furthermore, for the sake of brevity, the parity plots from all the different trials are not shown here. Nevertheless, the parity plot for one selected trial is sufficient to demonstrate the benefits of HPO. In all the parity plots shown, the dotted line represents the regression line that would occur in a situation with zero ML bias, while the solid line shows the actual regression line. The parity plots for ISFC based on case A and case C presented in Fig. 3 results show that AutoML-GA leads to much better agreement with the actual values of ISFC compared to ML-GA.

In general, the level of scatter from the ideal regression line is much lower, with higher R^2 values across all the base learners. The improvement is most notable in XGB and KRR, showing that the default values for the base learners were especially poor choices. From the super learner predictions of case A and case C in Fig. 3, it can be seen that AutoML-GA produces significantly less scatter with a higher R^2 value of 0.96 compared to ML-GA with 0.919. Also, it can be seen that by reducing the errors in the outliers, the super learner predictions have lower scatter and higher R^2 values than their base learner counterparts, as expected. The R^2 values attained by the base learners for all target variables are shown in Table 6.

Figure 4 shows the mean squared error (MSE) for four selected intermediate output variables (ISFC, SOOT, NO_x, and MPRR). The heights of the bars in the bar chart represent the averaged mean square error (AMSE) across different trials, normalized by the mean of the respective variables. The error bars represent one standard deviation. Shorter bars represent lower variances in the errors across the ten different trials run. The charts show that in general, for all output variables, the AutoML-GA leads to much lower MSE compared to ML-GA. For instance, for MPRR, the errors are reduced by 29%, 42%, 49%, 83%, and 50% for ANN, SVR, RPR, KRR, and XGB, respectively. The super learner errors on average are reduced by 47%, 46%, 64%, and 35% for ISFC, SOOT, NO_x, and MPRR, respectively. Aside from the reduction in the mean, case C also shows smaller error bars, signifying higher confidence in the MSE obtained from the optimized base learners.

In Fig. 5, maximum merit values obtained from cases A, B, and C are shown as functions of the number of iterations. In Figs. 5(*a*)–5(*c*), the individual runs are shown in gray lines along with the average maximum merit across all the cases. In all cases, the design optima obtained using the ML models are evaluated using the ground-truth function at each iteration. Therefore, the merit values shown in the plots are not the predicted merit values obtained from the different approaches, but rather, the actual merit values based on the ground-truth function. The huge variation in the trajectories of the gray lines shows the high degree of variance that can occur over multiple trials. For example, for case B, the best trial reaches a merit of 104.0 in three iterations, while in the worst scenario it does not reach 104.0 until the 19th iteration. In Fig. 5(*d*), the maximum merit as a function of the number of iterations is

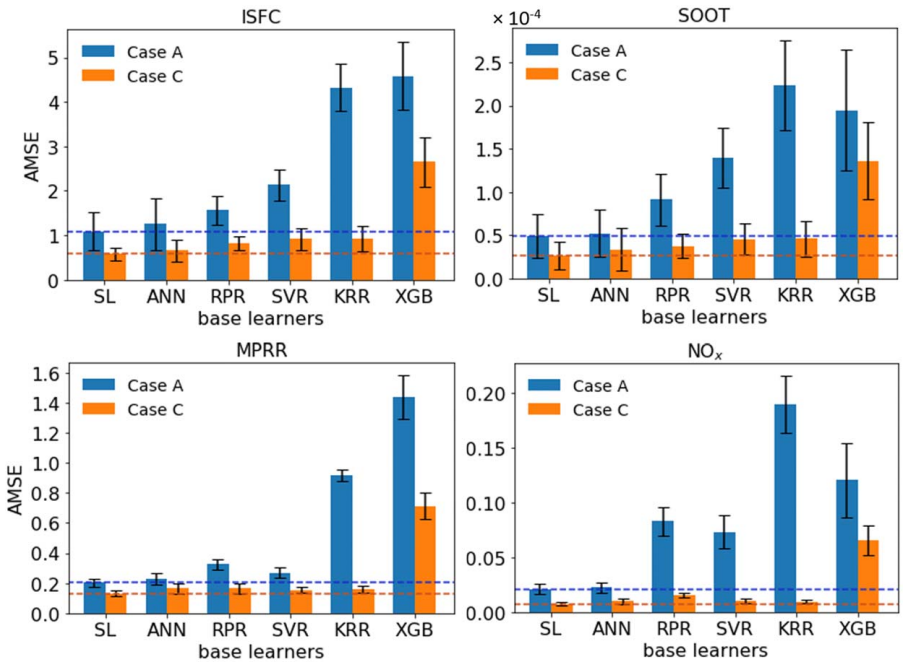

Fig. 4 Comparison of case A and case C AMSE obtained using the super learner and various base learners. The AMSE shown in the plots are normalized by the means of the respective parameters.

Downloaded from http://asmedigitalcollection.asme.org/energyresources/article-pdf/143/8/082305/6680013/jert_143_8_082305.pdf by Purdue University at West Lafayette user on 17 April 2021

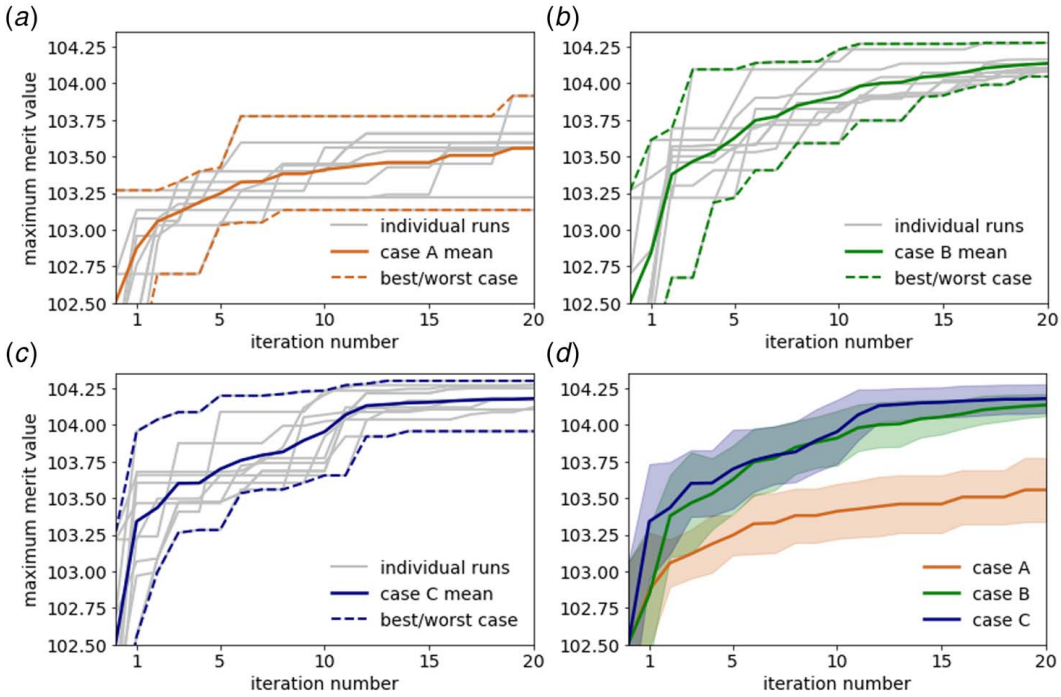

Fig. 5 Maximum merit value as a function of iteration number for cases A, B, and C

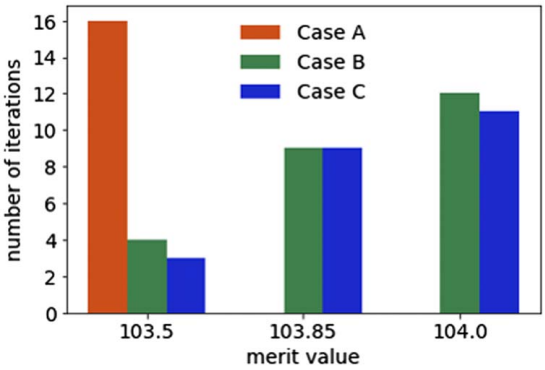

Fig. 6 Number of iterations required to reach various merit values

compared for all cases. The shaded regions in the plots correspond to one standard deviation, while the solid lines are the merit values averaged over all trials. The plots show that for a given number of design iterations, cases B and C are able to achieve much higher merits compared to case A. After 20 iterations are completed, the mean merit values attained are 103.56, 104.14, and 104.18 for cases A, B, and C, respectively.

Figure 6 shows the number of iterations taken for the average merit value to reach a specified merit. Three levels of merit are shown: 103.5, 103.85, and 104.0. Because of the small starting dataset of size 150 used in this study, case A is unable to reach the latter two thresholds of 103.85 and 104.0 in 20 iterations, taking as much as 16 iterations to reach 103.5. It can be seen that while the case C appears to converge slightly faster than case B, both cases are similar, suggesting that the cost of additional HPO every five iterations may not be worth the benefit. Indeed, the differences between both cases are within the reasonable limits of

Table 7 Optimum hyperparameters for different base models and target variables

	ISFC	SOOT	NO_x	MPRR	PMAX
RPR					
Regularization weight, α	0.03314	0.0423	0.0032	0.0172	0.0028
Polynomial degree	3	4	2	3	3
SVR					
Kernel coefficient, γ	0.0983	0.0731	0.0950	0.1637	0.0449
Parameter to control the number of support vectors, υ	0.6953	0.5503	0.9060	0.7080	0.6200
Penalty parameter, C	138.1	211	107.9	56.53	280.5
KRR					
Gamma parameter, γ	0.0512	0.0324	0.0763	0.12	0.0295
Regularization parameter, α	4.65×10^{-5}	1.68×10^{-5}	5.31×10^{-5}	9.39×10^{-4}	5.70×10^{-6}
XGB					
Number of trees	3674	5910	5036	5456	6120
Maximum tree depth	2	3	2	2	2
Learning rate	0.1295	0.08374	0.05865	0.0567	0.04312
ANN					
Number of neurons in each hidden layer	187	185	224	176	213
Tolerance to determine if training should be stopped early, tol	3.20×10^{-6}	5.30×10^{-6}	1.13×10^{-6}	1.97×10^{-6}	1.60×10^{-6}
L2 regularization parameter, α	3.78×10^{-3}	2.93×10^{-3}	4.27×10^{-3}	3.93×10^{-3}	1.97×10^{-3}

Downloaded from http://asmedigitalcollection.asme.org/energyresources/article-pdf/143/8/082305/6680013/jert_143_8_082305.pdf by Purdue University at West Lafayette user on 17 April 2021

uncertainty that exist with the limited number of trials that were run, as seen in the standard deviation of the maximum merit in Fig. 5(*c*). However, differences between cases B and C could be higher depending on the type of optimization problem and size of the design space. Overall, the results show that AutoML-GA can provide superior prediction accuracy, and hence better merit values compared to an ML-GA approach that utilizes default hyperparameters.

Lastly, the optimal hyperparameters obtained for all cases are shown in Table 7. Two conclusions can be drawn from the table. First, a comparison of the optimized hyperparameters and the default hyperparameters (listed in Table 1) reveals very marked differences. This shows that for a given problem, the default hyperparameters can be very far from the best hyperparameters. Second, while there are some exceptions, there is generally considerable variation in the selected hyperparameters for the different target variables of interest. The second point further demonstrates the importance of automated hyperparameter tuning for engine design optimization, since manually selecting hyperparameters for each algorithm for all target variables is practically infeasible. Furthermore, by comparing the selected hyperparameters in Table 7 with the default hyperparameters, an understanding of why the selected hyperparameters are more effective can be deduced. For instance, for RPR, the selected regularization weight is weaker while the degree of the polynomial is generally higher when compared to the default values. This suggests that the default hyperparameters significantly underfit and shows that the selected hyperparameters are effective because they allow for greater fitting power. On the other hand, XGB shows a lower tree depth and a larger number of trees for the optimized hyperparameters. This suggests that the chosen hyperparameters are effective because they correct the default model's tendency to overfit.

5 Conclusions

In this work, an automated ML approach for design optimization, AutoML-GA, was introduced. The framework eliminates trial-and-error and the need for a priori knowledge when developing surrogate-based optimization tools, by incorporating an automated selection of best hyperparameters. The proposed approach was tested on an optimization problem corresponding to IC engine combustion case, where the goal was to minimize the ISFC while respecting certain constraints. An iterative loop was developed where the resulting global optimum was added to the training data, and the optimization process was repeated for a total of 20 design iterations. The results showed marked improvements in the training errors and the design optimum obtained for the cases where HPO was performed. ML-GA approach using default hyperparameters failed to reach a good merit value even after 20 iterations were completed. The average improvement in merit was 1.02% for ML-GA, and 1.59% for the AutoML-GA with HPO performed once before the first iteration. Performing AutoML-GA with HPO repeatedly done every five iterations yielded an improvement of 1.63%. Future work will involve coupling the AutoML-GA framework to CFD, investigating the influence of the size of the initial training data and exploring techniques such as asynchronous batch Bayesian optimization to reduce the overhead cost of AutoML-GA.

Acknowledgment

This paper has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory (Argonne). This work was supported by the U.S. Department of Energy, Office of Science under contract DE-AC02-06CH11357. The research work was funded by the Department of Energy Technology Commercialization Fund (TCF) project. The authors also acknowledge the computing resources provided on "Blues" and bebop, high-performance computing clusters operated by the Laboratory Computing Resource Center (LCRC) at Argonne National Laboratory.

Conflict of Interest

There are no conflicts of interest.

Data Availability Statement

The datasets generated and supporting the findings of this article are obtained from the corresponding author upon reasonable request. The authors attest that all data for this study are included in the paper.

References

[1] Broatch, A., Novella, R., Gomez-Soriano, J., Pal, P., and Som, S., 2018, "Numerical Methodology for Optimization of Compression-Ignited Engines Considering Combustion Noise Control," SAE Int. J. Engines, **11**(6), pp. 625–642.
[2] Ge, H.-W., Shi, Y., Reitz, R. D., Wickman, D. D., and Willems, W., 2009, "Optimization of a HSDI Diesel Engine for Passenger Cars Using a Multi-Objective Genetic Algorithm and Multi-Dimensional Modeling," SAE Int. J. Engines, **2**(1), pp. 691–713.
[3] Hiroyasu, T., Miki, M., Kamiura, J., Watanabe, S., and Hiroyasu, H., 2002, "Multi-Objective Optimization of Diesel Engine Emissions and Fuel Economy Using Genetic Algorithms and Phenomenological Model," No. 0148-7191, SAE Technical Paper.
[4] Magnier, L., and Haghighat, F., 2010, "Multiobjective Optimization of Building Design Using TRNSYS Simulations, Genetic Algorithm, and Artificial Neural Network," Build. Environ., **45**(3), pp. 739–746.
[5] Wickman, D. D., Senecal, P. K., and Reitz, R. D., 2001, "Diesel Engine Combustion Chamber Geometry Optimization Using Genetic Algorithms and Multi-Dimensional Spray and Combustion Modeling," SAE Trans., pp. 487–507.
[6] Lu, Y., Li, J., Xiong, L., and Li, B., 2019, "Simulation and Experimental Study of a Diesel Engine Based on an Electro-Hydraulic FVVA System Optimization," ASME J. Energy Resour. Technol., **142**(3), p. 032204.
[7] Hamel, J. M., Allphin, D., and Elroy, J., 2018, "Multi-Objective Optimization Model Development to Support Sizing Decisions for a Novel Reciprocating Steam Engine Technology," ASME J. Energy Resour. Technol., **140**(7), p. 072204.
[8] Probst, D. M., Senecal, P. K., Chien, P. Z., Xu, M. X., and Leyde, B. P., 2018, "Optimization and Uncertainty Analysis of a Diesel Engine Operating Point Using Computational Fluid Dynamics," ASME J. Eng. Gas Turbines Power, **140**(10), p. 102806.
[9] Pei, Y., Pal, P., Zhang, Y., Traver, M., Cleary, D., Futterer, C., Brenner, M., Probst, D., and Som, S., 2019, "CFD-Guided Combustion System Optimization of a Gasoline Range Fuel in a Heavy-Duty Compression Ignition Engine Using Automatic Piston Geometry Generation and a Supercomputer," SAE Int. J. Adv. Curr. Pract. Mob., **1**, pp. 166–179.
[10] Ashok, B., Jeevanantham, A. K., Prabu, K., Shirude, P. M., Shinde, D. D., Nadgauda, N. S., and Karthick, C., 2020, "Multi-Objective Optimization on Vibration and Noise Characteristics of Light Duty Biofuel Powered Engine at Idling Condition Using Response Surface Methodology," ASME J. Energy Resour. Technol., **143**(4), p. 042301.
[11] Marri, V. B., Madhu Murthy, K., and Amba Prasad Rao, G., 2020, "Optimization of Operating Parameters of an Off-Road Automotive Diesel Engine Running at Highway Drive Conditions Using Response Surface Methodology," ASME J. Energy Resour. Technol., pp. 1–48.
[12] Bertram, A. M., and Kong, S.-C., 2019, "Computational Optimization of a Diesel Engine Calibration Using a Novel SVM-PSO Method," No. 0148-7191, SAE Technical Paper.
[13] Karra, P., and Kong, S.-C., 2010, "Application of Particle Swarm Optimization for Diesel Engine Performance Optimization," No. 0148-7191, SAE Technical Paper.
[14] Zhang, Q., Ogren, R. M., and Kong, S.-C., 2017, "Application of Improved Artificial Bee Colony Algorithm to the Parameter Optimization of a Diesel Engine With Pilot Fuel Injections," ASME J. Eng. Gas Turbines Power, **139**(11), p. 112801.
[15] Kennedy, J., and Eberhart, R., 1995, "Particle Swarm Optimization," Proceedings of the IEEE International Conference on Neural Networks, Perth, WA, Australia, Nov. 27–Dec. 1.
[16] Karaboga, D., and Basturk, B., 2007, "A Powerful and Efficient Algorithm for Numerical Function Optimization: Artificial Bee Colony (ABC) Algorithm," J. Global Optim., **39**(3), pp. 459–471.
[17] Kavuri, C., and Kokjohn, S. L., 2018, "Exploring the Potential of Machine Learning in Reducing the Computational Time/Expense and Improving the Reliability of Engine Optimization Studies," Int. J. Engine Res., **21**(7), pp. 1251–1270.
[18] Owoyele, O., and Pal, P., 2020, "A Novel Active Optimization Approach for Rapid and Efficient Design Space Exploration Using Ensemble Machine Learning," ASME J. Energy Resour. Technol., **143**(3), p. 032307.
[19] Owoyele, O., and Pal, P., 2021, "A Novel Machine Learning-Based Optimization Algorithm (ActivO) for Accelerating Simulation-Driven Engine Design," Appl. Energy, **285**, p. 116455.
[20] Moiz, A. A., Pal, P., Probst, D., Pei, Y., Zhang, Y., Som, S., and Kodavasal, J., 2018, "A Machine Learning-Genetic Algorithm (ML-GA) Approach for Rapid

Downloaded from http://asmedigitalcollection.asme.org/energyresources/article-pdf/143/8/082305/6680013/jert_143_8_082305.pdf by Purdue University at West Lafayette user on 17 April 2021

Optimization Using High-Performance Computing," SAE Int. J. Commer. Veh., **11**(5), pp. 291–306.
[21] Badra, J., Sim, J., Pei, Y., Viollet, Y., Pal, P., Futterer, C., Brenner, M., Som, S., Farooq, A., and Chang, J., 2020, "Combustion System Optimization of a Light-Duty GCI Engine Using CFD and Machine Learning," No. 0148-7191, SAE Technical Paper.
[22] Badra, J. A., Khaled, F., Tang, M., Pei, Y., Kodavasal, J., Pal, P., Owoyele, O., Fuetterer, C., Brenner, M., and Farooq, A., 2020, "Engine Combustion System Optimization Using CFD and Machine Learning: A Methodological Approach," ASME J. Energy Resour. Technol., **143**(2), p. 022306.
[23] Probst, D. M., Raju, M., Senecal, P. K., Kodavasal, J., Pal, P., Som, S., Moiz, A. A., and Pei, Y., 2019, "Evaluating Optimization Strategies for Engine Simulations Using Machine Learning Emulators," ASME J. Eng. Gas Turbines Power, **141**(9), p. 091011.
[24] Van der Laan, M. J., Polley, E. C., and Hubbard, A. E., 2007, "Super Learner," Stat. Appl. Genet. Mol. Biol., **6**(1).
[25] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., and Dubourg, V., 2011, "Scikit-Learn: Machine Learning in Python," J. Mach. Learn. Res., **12**(Oct), pp. 2825–2830.
[26] Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., and van der Walt, S. J., 2020, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," Nat. Methods, **17**(3), pp. 261–272.
[27] Van Der Walt, S., Colbert, S. C., and Varoquaux, G., 2011, "The NumPy Array: A Structure for Efficient Numerical Computation," Comput. Sci. Eng., **13**(2), pp. 22–30.
[28] Drucker, H., Burges, C. J., Kaufman, L., Smola, A. J., and Vapnik, V., 1996, "Support Vector Regression Machines," Proc. Adv. Neural Inform. Process. Syst., pp. 155–161.
[29] Chen, T., and Guestrin, C., 2016, "Xgboost: A Scalable Tree Boosting System," Proceedings of the 22nd acm Sigkdd International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, Aug. 13–17, pp. 785–794.
[30] Mockus, J., 2012, *Bayesian Approach to Global Optimization: Theory and Applications*, Springer Science & Business Media, The Netherlands.
[31] Močkus, J., 1974, "On Bayesian Methods for Seeking the Extremum," Proceedings of the Optimization Techniques IFIP Technical Conference, Novosibirsk, Russia, July 1–7, Springer, pp. 400–404.
[32] Nogueira, F., 2014, "Bayesian Optimization: Open Source Constrained Global Optimization Tool for Python."
[33] Richards, K., Senecal, P., and Pomraning, E., 2019, *CONVERGE 2.3*, Convergent Science, Madison, WI.
[34] Pal, P., Probst, D., Pei, Y., Zhang, Y., Traver, M., Cleary, D., and Som, S., 2017, "Numerical Investigation of a Gasoline-Like Fuel in a Heavy-Duty Compression Ignition Engine Using Global Sensitivity Analysis," SAE Int. J. Fuels Lubr., **10**(1), pp. 56–68.

Downloaded from http://asmedigitalcollection.asme.org/energyresources/article-pdf/143/8/082305/6680013/jert_143_8_082305.pdf by Purdue University at West Lafayette user on 17 April 2021