**Proceedings of the ASME 2023**
**ICE Forward Conference**
**ICEF2023**
**October 8-11, 2023, Pittsburgh, Pennsylvania**

**ICEF2023-110524**

# A DEEP LEARNING APPROACH TO PREDICT IN-CYLINDER PRESSURE OF A COMPRESSION IGNITION ENGINE

**Rodrigo Ristow Hadlich**[1,2,3], **Jason Loprete**[1,2,3], **Dimitris Assanis**[1,2,3,*],

[1]Department of Mechanical Engineering, Stony Brook University, Stony Brook, NY
[2]Institute for Advanced Computational Science, Stony Brook, NY
[3]Advanced Energy Research and Technology Center, Stony Brook, NY

## ABSTRACT

*As emissions regulations for greenhouse gas emissions become more strict, it is important to increase the efficiency of engines by improving on the design and operation. Current optimization methods involve performing large numbers of experimental investigations on physical engines or making use of detailed Computational Fluid Dynamics modeling efforts to provide visual and statistical insights on in-cylinder behavior. The latter still requires experimental data for model validation. Both of these methods share a common set of problems, that of being monetarily expensive and time consuming. Previous work has proposed an alternative method for engine optimization using machine learning (ML) models and experimental validation data to predict scalar values representing different parameters. With such models developed, one can then quickly iterate on operating conditions to find the point that maximizes an application-dependent reward function. While these ML methods provide information on individual performance parameters, they lack key information of in-cylinder indicators such as cylinder pressure traces and heat release curves that are traditionally used for performance analysis. This work details the process of implementing a Multilayer Perceptron (MLP) model capable of accurately predicting crank-angle resolved high-speed in-cylinder pressure using equivalence ratio, fuel injection pressure and injection timing as input features. It was demonstrated that the model was able to approximate engine behavior with mean squared error lower than 0.05 on a 1-55 range in the test set. This approach shows potential for greatly accelerating the optimization process in engine applications.*

## NOMENCLATURE

*Abbreviations*

| | |
|---|---|
| aTDC | After Top Dead Center |
| CAD | Crank Angle Degrees |
| COV | Coefficient of Variation |
| EVC | Exhaust Valve Closing |
| EVO | Exhaust Valve Opening |
| HPO | Hyperparameter Optimization |
| IMEP | Indicated Mean Effective Pressure |
| IVC | Intake Valve Closing |
| IVO | Intake Valve Opening |
| MAE | Mean Absolute Error |
| ML | Machine Learning |
| MLP | Multilayer Perceptron |
| MSE | Mean Squared Error |
| ReLU | Rectified Linear Unit |
| RPM | Rotations per Minute |
| TDC | Top Dead Center |

*Greek letters*

| | |
|---|---|
| $\phi$ | Equivalence Ratio |

*Superscripts and subscripts*

| | |
|---|---|
| n | Net |

*Corresponding author: dimitris.assanis@stonybrook.edu

## 1. INTRODUCTION

With the imminent threat of global warming due to the excess of greenhouse gases in the atmosphere, emissions regulations are becoming stricter. Although the percentage of electric vehicles on the road is increasing, engines are still expected to be largely present in the transportation sector in the near future. In particular compression ignition engines, which are mostly used in heavy-duty vehicles that tend to have high load and long range requirements, and are thus difficult to electrify. In order to keep up with the changing regulations, engines must be optimized to convert energy efficiently while keeping emissions low.

The most reliable method of engine optimization is to perform physical experiments and tune parameters to set of conditions that optimizes a given output parameter or reward function. However, it is expensive and time consuming since it requires a great deal of trial and error and experiments take long to reach steady state operation at each operating condition. One way to improve on this optimization process is to have numerical models of the engine that can quickly iterate on the operating conditions and generate outputs that closely agree with experimental data. Traditional engine models involve zero-dimensional [1–3], one-dimensional [4], and three-dimensional [5–8] computational models that take into account varying levels of detail about the physics of the engine breathing events as well as the combustion process. However, each of these presents drawbacks when applied in engine optimization. One of the shortcomings in all of these models is the time and effort required to properly tune the model so that it agrees with experimental results, which can be quite significant. In addition, 3D Computational Fluid Dynamics models are also extremely expensive when it comes to computational resources, and thus become even more expensive and time consuming than physical experiments. Reduced Order Modeling methods have also been applied in previous work [9–11] to aid physics models in predicting engine behavior, but suffer from an extensive calibration process. A reduced order modeling method was also compared to Machine Learning (ML) algorithms in [12], where the ML based algorithm showed significant improvement in performance.

Considering the shortcomings of physics-based methods, the applications of ML algorithms to study engine control and modeling [13] as well as general combustion science [14] and chemistry [15] have been extensively studied in recent years. In engine modeling, the principle of ML methods is that they operate in a data-driven black-box approach, where the input/output relationship from experimental data is non-linear and multi-dimensional, and the ML model is used to curve-fit in this multi-dimensional space. This is appealing from an optimization perspective because if a (relatively) small dataset is generated across a wide range of operating conditions, the ML model can learn from this data and quickly predict engine behavior at any point inside the sampled range and thus find optimal operating conditions. Many attempts at predicting engine behavior with ML models have been successful, such as predicting exhaust gas temperature [16], brake power [17], torque [17, 18], brake specific fuel consumption [17–19], emissions concentrations [17, 19–21], engine efficiency [19], combustion timing of a Homogeneous Charge Compression Ignition (HCCI) engine [22], and exergetic parameters [23].

There have been efforts in existing literature to predict engine in-cylinder pressure using Extreme Learning Machine [24]. The model detailed in [24] used the crankshaft position and instantaneous engine speed as inputs to the model, and for every combination of crankshaft position and engine speed it predicts a value for the in-cylinder pressure. However, such a model would only work to quantify the in-cylinder pressure for an engine with previously tuned and fixed parameters in the Engine Control Unit, meaning at a given engine speed all parameters would be constant independent of throttle position. While it would allow for having on-board access to the in-cylinder pressure in a vehicle on the road, it would not be useful in the engine research and optimization process. To the best of the authors' knowledge, there are currently no deep learning models that can predict crank-angle resolved in-cylinder pressure while taking into account many operating conditions to make the model more flexible and allow for a broader exploration process.

Once a model that can reliably and quickly predict engine behavior is obtained, it can be coupled with numerical optimization algorithms to find the operating conditions that maximize a reward function. This reward function can be specified by the user depending on the application. ML methods are great candidates to be used with such algorithms due to its high accuracy and low computational cost, as demonstrated in literature. Examples include using this pathway to optimize engine geometry [25–30] as well as engine operating parameters [25, 29, 31].

Therefore, in this work a Multilayer Perceptron (MLP) model was developed to predict crank-angle resolved in-cylinder pressure values for internal combustion engines, in this case a compression ignition engine, using as input the equivalence ratio, fuel injection timing and fuel injection pressure. Experimental data was collected in-house and used for training and validation of the model. Hyperparameter Optimization (HPO) was performed to find the optimal hyperparameters for the model.

## 2. METHODOLOGY

The following section provides details about the data collection process, as well as the hardware and software used, and insights on the parameters of the model, the model optimization process and data pre-processing.

### 2.1 Experimental Setup and Procedure

In order to properly train and test the model, a symmetric and consistent data set is ideal. The data used in this work was collected in-house using a single-cylinder compression ignition Ricardo Hydra research engine. Details of the engine geometry are provided in Table 1. The in-cylinder pressure was measured every 0.1 Crank Angle Degrees (CAD) using a Kistler 6045B pressure transducer. The fuel injection pressure is measured by a Kistler 4067E pressure sensor also at a resolution of 0.1 CAD. The actuator to control the fuel pressure is a Bosch CP3 injection pump, and the controls for the sensor-actuator fuel system are implemented in an in-house LABVIEW code. The LABVIEW code is used to display relevant information from raw as well as processed data from the sensors and also to communicate user inputs to the actuators. The equivalence ratio ($\phi$) of the combustion is measured in the exhaust runner by a LambdaCAN, and it is controlled by manually setting the desired opening time of the fuel injector in the LABVIEW code. A DyneSystems eddy current dynamometer is connected to the engine's crankshaft via a driveshaft and is used to measure the torque output as well as to control the engine speed according to the desired user input.

Throughout the experiments the engine speed was held constant at 1200 rotations per minute (RPM). In order to protect engine parts from damage due to excessive pressure increase, the knock limit was established to be when the maximum pressure rise rate (MPRR) exceeded 10 bar/CAD, so only data points that fell below this threshold were recorded. The procedure adopted for the experiments was to first set a desired fuel injection pressure and $\phi$. Once the engine was operating at these conditions, a fuel injection timing sweep was performed where the injection timing was varied from knock limit to misfire limit, with the step size varying depending on how wide the range of possible injection timings between knock and misfire limits was. Once the injection timing was swept at a given $\phi$ and fuel injection pressure, the $\phi$

**TABLE 1: EXPERIMENTAL SETUP DETAILS**

| | |
|---|---|
| Stroke [mm] | 86 |
| Bore [mm] | 79 |
| Connecting Length Rod [mm] | 160 |
| Compression Ratio | 15.5:1 |
| Number of Valves per Cylinder | 4 |
| Piston Pin Offset [mm] | 0.6 |
| Exhaust Valve Opening (EVO) [°aTDC] | 122 |
| Exhaust Valve Closing (EVC) [°aTDC] | 366 |
| Intake Valve Opening (IVO) [°aTDC] | -354 |
| Intake Valve Closing (IVC) [°aTDC] | -146 |

was increased and the same process of injection timing sweep was repeated. The values of $\phi$ were varied from 0.2 to 0.4 in steps of 0.05. Once all injection timings and all $\phi$ values were explored at a given injection pressure, the injection pressure was increased and the same injection timing and $\phi$ sweeps were performed. The fuel injection pressure was varied from 450 bar to 850 bar in steps of 100 bar. For each point in this exploration matrix two files were saved, each containing the values for 300 consecutive cycles at a fixed condition (within experimental error). All parameters varied in the experiments, namely equivalence ratio, fuel injection pressure, and fuel injection timing, were used as inputs features to the model, with the cylinder pressure as the output.

## 2.2 Computational Setup

The computer used for the model training and validation was a Dell Precision 3660 with a 12$^{th}$ Gen Intel(R) Core i7-12700 Central Processing Unit and NVIDIA RTX A2000 Graphics Processing Unit (GPU). The programming language of choice is Python. Data pre-processing was done using the Python library Numpy [32], and the ML models were built using the deep learning library PyTorch [33]. For the HPO process the library Optuna [34] was used.

## 2.3 Model Architecture and Parameter Selection

Multilayer Perceptrons (MLP), also known as Feed Forward Neural Networks, are the most basic of the deep learning architectures. MLP models have in general three types of layers: an input layer, one or more hidden layers, and an output layer. As the name suggests, the input layer contains the input data that is being used to make a prediction. Each node in the input layer represents one feature, or one parameter that is to be used by the model to make the prediction. The hidden layer, or sequence of them, is the heart of the model in which the data transformations are performed. The goal of the hidden layers is to change the representation of the data so that it can be linearly separable at the output layer. Lastly, the output layer is where the predicted values are outputted by the model. Each node in the output layer represents one parameter that is to be predicted by the model. Moving from the input layer in the direction of the output layer is known as a forward pass through the model.

In MLP models, each node from one layer is connected to every node in the next layer, and is not connected to nodes in its own layer. This connection, or edge, is given by an affine

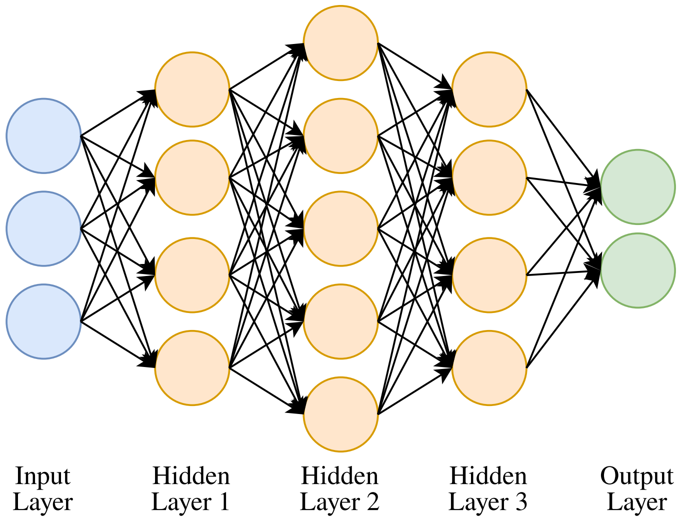


**FIGURE 1: MULTILAYER PERCEPTRON MODEL ILLUSTRATION**

transformation followed by a non-linear activation function. The vector of values at hidden layer $i$, labeled $\mathbf{h}_i$, can be calculated as follows:

$$\mathbf{h}_i = g_i(\mathbf{A}_i \mathbf{h}_{i-1} + \mathbf{b}_i) \quad (1)$$

Where $\mathbf{A}_i$ is the matrix containing the weights of all the edges connecting layer $i-1$ to layer $i$, $\mathbf{h}_{i-1}$ is the vector containing the values at each node of hidden layer $i-1$, $\mathbf{b}_i$ is the bias vector for hidden layer $i$, and $g_i(\cdot)$ is the non-linear activation function for layer $i$. The weight matrix $\mathbf{A}_i$ and the bias vector $\mathbf{b}_i$ are the trainable parameters of layer $i$, and the combination of weights and biases is typically referred to as simply the weights of the model. Figure 1 shows a diagram of an MLP model with three input parameters, three hidden layers, and two output parameters. Each arrow in the diagram represents an edge connecting the previous layer to the next layer, meaning Eqn. 1 is applied to the data from the node in the previous layer to yield the value in the node in the following layer.

The error between the true label $\mathbf{y}$ and the predicted values $\hat{\mathbf{y}}$, both of which have the same dimensions and are vectors of the same length as the number of parameters being predicted, is given by the loss function. In order to train the weights and biases of all the layers in the model to reduce the value of the loss function, and thus reduce the error in the predicted values, the gradient of the loss function with respect to the weights must be calculated. For this, the backpropagation algorithm is used [35], which calculates the gradient of the loss function by making a backward pass in the computation graph given by the model. Once the gradient is calculated, an optimization algorithm must be employed to update the weights using this gradient. Moving from the output layer to the input layer to calculate the gradients and update the weights of the model is known as a backward pass through the model.

Deep learning models such as MLPs have many hyperparameters. These are parameters that affect the performance of the model and that must be chosen by the creator of the model. The values of hyperparameters that optimize the performance of a model tend to vary significantly depending on many factors such as whether it is a regression or classification problem, how much training data is available, how easily patterns can be observed in the data, etc. Examples of hyperparameters include, but are not limited to, the number of nodes in each hidden layer, the number of hidden layers, the activation function, the loss function, and the algorithm used for optimization. A common way of finding good values for the hyperparameters for a specific problem is to employ some form of HPO algorithm, such as grid search, to test many different values and choices of functions and evaluate which combination produces the best result.

**2.3.1 Loss Function and Evaluation Metrics.** In order for the model to minimize the difference between true and predicted values, a loss function must be selected to quantify the difference between true and predicted vectors in a single scalar value. Two of the most widely used loss functions for a regression problem such as the one presented in this work are the Mean Absolute Error (MAE) and Mean Squared Error (MSE). The formulas for calculating each of these errors are listed below.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\mathbf{y}_i - \hat{\mathbf{y}}_i| \tag{2}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{3}$$

Where $y$ represents the vector containing the true values, $\hat{y}$ represents the vector containing the predicted values, and $n$ stands for the number of values in $y$ and $\hat{y}$.

Because the MSE increases with the square of the difference between true and predicted values, it penalizes the model more heavily for larger discrepancies in the predicted value than the MAE. Due to the large fluctuation in the magnitude of the pressure values in each cycle, ranging from 1 to as high as 80, the loss function should be aggressive in penalizing large differences so that the model can more easily approximate the shape of an engine pressure trace. For this reason, the MSE loss function was chosen for this application.

While the MSE can also be used as a performance metric, it is important to quantify the "goodness" of the fit in more than one way. For this purpose, the coefficient of determination, or $R^2$ score, is used. This is a tool that evaluates the linear correlation between the predicted value $\hat{y}$ and the true value $y$. It is usually a value between 0 and 1, with 1 indicating a strong linear correlation. It is calculated using the formula below.

$$R^2 = 1 - \frac{\sum_i(\mathbf{y}_i - \hat{\mathbf{y}}_i)^2}{\sum_i(\mathbf{y}_i - \bar{y})^2} \tag{4}$$

Where $\mathbf{y}$ is the vector containing the true values, $\hat{\mathbf{y}}$ is the vector containing the predicted values, and $\bar{y}$ is the mean of the vector $\mathbf{y}$. The values reported for the $R^2$ score throughout this work are calculated for each individual cycle, then averaged for the batch, and lastly averaged for the set. This means that for every cycle the $R^2$ score was calculated to analyze the linear correlation between the true values for that cycle and the predicted values. All the individual values for each cycle in a batch of examples were calculated and summed, then divided by the number of examples in the batch. Lastly, all the values for the $R^2$ scores of all batches were summed and divided by the number of batches in the set (either training, validation, or test set).

**2.3.2 Hyperparameters for Optimization.** To get the model to perform well, it is necessary to tune many hyperparameters simultaneously. That is the purpose of the HPO process, to train the model with different combinations of values and evaluate which yields the best performance. However, due to the iterative nature of HPO, it is a computationally intensive task. It was thus decided by the authors to carefully select which hyperparameters would be optimized during HPO and which would not, balancing a trade-off between computation time and estimated impact on model performance based on previous knowledge. The optimizer for updating the weights of the model was chosen to be Adam [36] in all cases, so it is not included in the HPO process. However, the Adam optimizer has its own set of hyperparameters, and the learning rate was chosen as a tunable parameter.

In this work, a common technique in ML called learning rate decay was employed. The idea behind this technique is to change the value of the learning rate of the optimizer as training progresses. This is beneficial because at the start of the training process there is a benefit to having a large learning rate so it can explore a wide range of values for the weights of the model, moving between many regions of the multidimensional space formed by the model's weights and explore many areas with different local optima. However, as the model advances in the training process, it should find its way to the general area with the local optimum that yields the lowest error in the prediction, ideally the global optimum. But with a large learning rate it will likely not converge to the local optimum and will instead move around it without ever reaching it. At this point the learning rate should be decreased so it can perform a more refined search for the optimum. There are many different strategies for how to update the learning rate, and a very common one to use is the Step Decay (StepLR Scheduler in PyTorch) which drops the learning rate by a given factor ($\gamma$) every given number of epochs, referred to as the step size. In all training processes in this work the Step Decay was used with a learning rate multiplicative factor $\gamma$ of 0.1 at a step size of 50 epochs.

Also on the list of hyperparameters in the HPO process are the number of hidden layers, the number of nodes in the hidden layers, and the choice of activation function between Rectified Linear Unit (ReLU) [37], leaky ReLU [38], and the hyperbolic tangent, tanh. The batch size, which is the number of training examples the model processes before updating the weights, was chosen to be fixed at 32. Table 2 summarizes the hyperparameters of the model, their range, and whether they were chosen to be optimized in the HPO process or not. The learning rate was sampled on a logarithmic scale between the ranges specified in Table 2, while the number of hidden layers was sampled linearly. As for the number of nodes in the hidden layers, a common practice is to use a number of nodes that is a power of 2. For this reason, the exponent $\mathbf{n}$ was sampled linearly and the number of nodes used for each sample was $2^{\mathbf{n}}$.

**TABLE 2: MODEL HYPERPARAMETERS**

| Hyperparameter | Range of Values | Optimized in HPO |
|---|---|---|
| Batch Size | 32 | No |
| Loss Function | MSE | No |
| Optimizer | Adam | No |
| $\beta_1, \beta_2, \epsilon$ [Adam] | 0.9, 0.999, $10^{-8}$ | No |
| Learning Rate | $[10^{-4}, 10^{-2}]$ | Yes |
| StepLR Scheduler $\gamma$ | 0.1 | No |
| StepLR Scheduler Step Size | 50 epochs | No |
| Number of Hidden Layers | [2, 10] | Yes |
| Nodes in Hidden Layers | $[2^7, 2^{11}]$ | Yes |
| Activation Function | [tanh, ReLU, Leaky ReLU] | Yes |

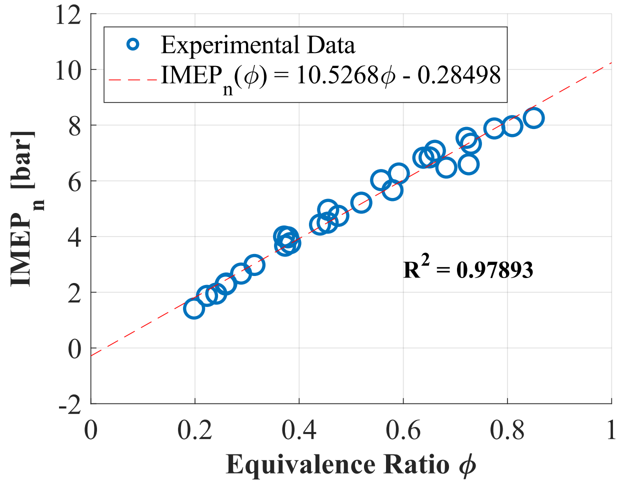


**FIGURE 2: $IMEP_n$ AS A FUNCTION OF $\phi$**

## 2.4 Data Description

Perhaps the most important part of developing a good machine learning model is understanding the data to be used. As mentioned in Section 2.1, the data was collected in-house in an experimental setup. For each set of operating conditions two data points were collected, each containing 300 individual cycles. When separating the full data set into training set, validation set, and test set, all 300 cycles corresponding to one data point were grouped together. Random sampling was then performed to sample which data points would be in each of the training, validation, and test sets. This ultimately means that if, for example, data point number 10 was sampled to be in the validation set, all 300 cycles corresponding to data point number 10 will be included in the validation set. Using this method, the full set which contains 192 data points, or 57600 cycles, was approximately divided into 80% for the training set, 10% for the validation set, and 10% for the test set. The sizes of the training, validation, and test sets used were 46500, 5400, and 5700, respectively.

For each cycle the parameter to be predicted, the cylinder pressure, contained 7200 values corresponding to the pressure value at every crank angle degree from -360° to 360°. However, it was found that the model was easily able to predict the behavior at intake and exhaust strokes. The most crucial part to be predicted was the combustion process, and for this reason it was decided to focus the model's efforts in predicting the pressure values only between -100 and 200 degrees relative to firing Top Dead Center (TDC). Therefore, the output layer of the model must have 3000 nodes, each corresponding to one pressure value between -100 and 200 degrees at a resolution of 0.1CAD. As for the input data, the three parameters that were varied in the experiments, namely fuel injection timing, fuel injection pressure of the cycle at time of injector opening, and equivalence ratio $\phi$, were used as inputs to the model as all three are known from previous knowledge of the system to have significant effect on the output.

However, available hardware for determining $\phi$ has a sampling rate lower than what is required to sample once per cycle at the engine speed of 1200RPM. In this setup it is measured once every two seconds, which is the equivalent of 15 measurements during 300 cycles or one measurement every 20 cycles. For this reason, feature engineering was required to make use of all 300 cycles for each data point given that the size of the data set contributes heavily to the performance of the model.

## 2.5 Data Pre-Processing

**2.5.1 Feature Engineering.** Because the $\phi$ is known from previous experiments to have a strong linear correlation to the net indicated mean effective pressure ($IMEP_n$), a statement can be made that:

$$\phi = f(IMEP_n) \tag{5}$$

The exact form of the relationship from Eq. 5 can be obtained by obtaining the slope of the curve generated by plotting $IMEP_n$ as a function of $\phi$. Given a group of data points, linear regression can be performed to obtain the equation of the line that minimizes the mean squared error between the line and the data points. Figure 2 shows the data obtained for this particular engine. Using the equation of the line from Fig. 2 and solving for $\phi$, Eq. 5 can be formally written as:

$$\phi = 0.095(IMEP_n) - 0.027 \tag{6}$$

The $IMEP_n$ has one value for every cycle since it is calculated from the cylinder pressure values according to the following equation:

$$IMEP_n = \frac{\int_{cycle} PdV}{V_d} \tag{7}$$

Where $P$ is the instantaneous in-cylinder pressure, $V$ is the instantaneous volume in the cylinder, and $V_d$ is the displacement volume of the engine. Another parameter that can be easily calculated is the Coefficient of Variation ($COV$) of the $IMEP_n$, labeled $COV_{IMEP_n}$, which is defined by the ratio of the standard deviation to the mean of the data set and measures the variability of a data set irrespective of the mean. And because of Eq. 5, it can be concluded that:

$$COV_{\phi} = f(COV_{IMEP_n}) \tag{8}$$

The actual form of Eq. 8 can be calculated from Eq. 6. Because the $COV$ quantifies the variation of a parameter, the $COV_{\phi}$ can be calculated by taking the first derivative of $\phi$ with respect to the $IMEP_n$, or the slope of the curve of Eq. 6.

$$COV_{\phi}(COV_{IMEP_n}) = \frac{d\phi}{dIMEP_n} = 0.095COV_{IMEP_n} \tag{9}$$

With this in mind, the following procedure was used to expand the $\phi$ measurements into a data set of the same length and same $COV$ as the $IMEP_n$ for each cycle. Because one $\phi$ measurement is made every 20 cycles, each data point containing 300 cycles is divided into groups of 20 cycles and each group has one $\phi$ measurement associated to it. Then, for each group, the offset coefficient for each of the 20 cycles is calculated based on the $IMEP_n$ using the following formula:

$$\delta = 0.095\frac{\mathbf{IMEP_n} - \mu_{IMEP_n}}{\mu_{IMEP_n}} \tag{10}$$

Where $\mathbf{IMEP_n}$ is the vector containing the $IMEP_n$ values and $\mu_{IMEP_n}$ is the mean of the $IMEP_n$. Using this offset coefficient, the vector containing the values of $\phi$ can then be calculated. It is assumed that the $\phi$ measurement associated to each group corresponds to the $\phi$ value of the first of the 20 cycles in the group. Therefore, the mean $\phi$ value of the group, $\mu_{\phi}$, can be calculated by:

$$\mu_{\phi} = \frac{\phi_{measured}}{1 + \boldsymbol{\delta}_0} \tag{11}$$

In Eqn. 11, $\phi_{measured}$ represents the measured $\phi$ value associated with the group and $\boldsymbol{\delta}_0$ represents the first $\delta$ value in the group. Finally, the vector of $\phi$ values corresponding to the group can be calculated by:

$$\boldsymbol{\phi} = \mu_{\phi}(1 + \boldsymbol{\delta}) \tag{12}$$

**2.5.2 Input Normalization.** Input normalization is widely used in machine learning because it is a simple process that can significantly speed up convergence during training. The idea behind it is to normalize all features of the model to have a mean of zero and be in the same range of values. This makes it so that the gradient is calculated with the same resolution for all features, and thus can converge at similar rates towards optima in all directions. There are many types of input normalization, and the one used in this work is described in the formula below:

$$\mathbf{X_{norm}} = \frac{\mathbf{X} - \boldsymbol{\mu}}{\boldsymbol{\sigma}} \tag{13}$$

Where $\mathbf{X_{norm}}$ is the normalized matrix containing the training set, $\mathbf{X}$ is the matrix containing the raw training set, $\boldsymbol{\mu}$ is the vector containing the mean of each feature of the training set, and $\boldsymbol{\sigma}$ is the vector containing the standard deviation of each feature of the training set. It is important to note that when evaluating the model with either the validation or test set, the input should be normalized using the same mean and standard deviation as the training set. The authors found that for the data set and model utilized in this work the use of input normalization improved convergence time by a factor of 4, and yielded 2 to 5 times lower MSE loss.

TABLE 3: HPO RESULTS

| Hyperparameter | Value |
|---|---|
| Learning Rate | $6.408 \cdot 10^{-4}$ |
| Number of Hidden Layers | 10 |
| Nodes in Hidden Layers | 256 |
| Activation Function | Leaky ReLU |

**2.5.3 Cylinder Pressure Filtering.** The raw cylinder pressure traces presented significant noise due to the inherent error of the physical sensor as well as traveling pressure zones in the combustion chamber. This noise was found to significantly degrade the performance of the model. To mitigate this, a second order Butterworth digital filter with critical frequency of 0.05Hz from the Scipy library was used.

## 3. RESULTS AND DISCUSSION

The following section provides details about the findings of the Hyperparameter Optimization process, and the final performance results obtained with the optimized model.

### 3.1 Model Optimization

The HPO process was performed using the Optuna [34] library. Optuna's default sampling algorithm is Tree Parzan Estimator [39], and this was the algorithm chosen for this work. The search space of the HPO is described in Sect. 2.3.2. During the HPO process, the loss function used was MSE for calculating the gradients and updating the weights of the model, and the metrics used to evaluate and compare the models was the coefficient of determination, or $R^2$ score. Each model was trained using the training data set and evaluated using the validation data set.

Two separate HPO studies were performed. Initially 2000 models with distinct combinations of hyperparameter values were evaluated to find the optimal combination. The number of epochs to train each model was set to 150; however, pruning was used to reduce the number of calculations. This is a feature offered in Optuna that monitors intermediate objective values and stops unpromising trials early in the training process. In this initial study it was found that the optimal number of layers for the model was of 2, which is the lowest value in the range tested. This behavior was unexpected seeing that models with larger number hidden of layers, or deeper models, should be able to generalize better to unseen data than shallow models with less hidden layers. Shallow models tend to overfit to the training set, and thus perform worse in unseen examples when compared to deeper models [40]. It was then proposed that perhaps shallow models would converge more quickly than deep models in the early stages of training due to this overfitting tendency. And because of pruning, the deep models were being ruled out as possible candidates due to slow convergence in the beginning, but would eventually achieve better performance if allowed to finalize the training process.

TABLE 4: PERFORMANCE METRICS FOR TRAIN, VALIDATION, AND TEST SETS

| Metrics | Train Set | Validation Set | Test Set |
|---|---|---|---|
| **MSE** | $3.0288 \cdot 10^{-2}$ | $5.9731 \cdot 10^{-2}$ | $4.3985 \cdot 10^{-2}$ |
| **R$^2$** | 0.99982 | 0.99964 | 0.99974 |

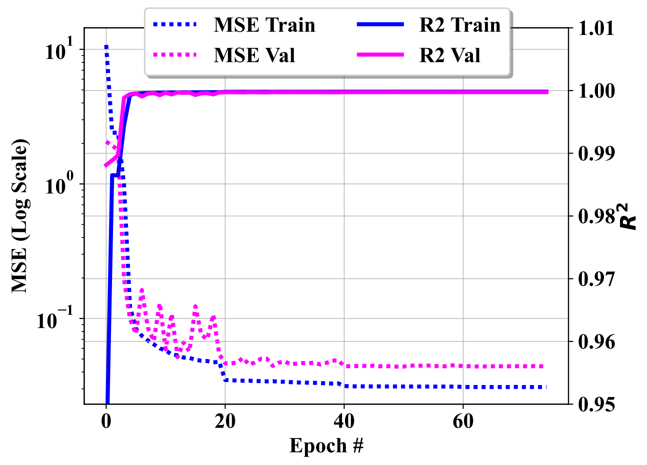


FIGURE 3: MSE LOSS AND $R^2$ SCORE OF TEST AND VALIDATION SETS DURING TRAINING OF MODEL

A second study was then performed without pruning and with 200 combinations of hyperparameters for 100 epochs. This study confirmed the thought proposed by showing the best performing model contained 10 hidden layers, the highest number of layers in the search space. Perhaps if more computational power was available it would have been demonstrated that the optimal number of hidden layers is even larger. The optimal combination of hyperparameter values obtained from the HPO process is listed in Table 3. However, possible overfitting was still observed in the model after HPO, and thus regularization was applied.

#### 3.1.1 Dropout Regularization.

Regularization techniques are commonly applied to prevent overfitting the model to the training data set. A simple yet effective regularization technique, known as dropout [41], works by dropping random nodes in each layer, as well as their connections to nodes in other layers, according to a probability $p$. The value of $p$ refers to the probability of a particular node being kept, thus a $p$ value of 1 would mean no nodes are dropped and $p = 0$ means every node is dropped. This essentially creates different architectures each time it samples from $p$ during training. During test time, dropout is not applied and the final result can be thought of as an average of all the architectures created during training. In this work, a dropout probability $p$ of 0.7 was used, and was found to prevent overfitting while not significantly reducing the training speed of the model. Due to the working principle of dropout, and following the practical guidance provided in [41], the number of nodes $n$ in each hidden layer was adjusted to be $n/p$. From the results of the HPO, $n = 256$, so the final number of nodes in each hidden layer reported in this work is 365.

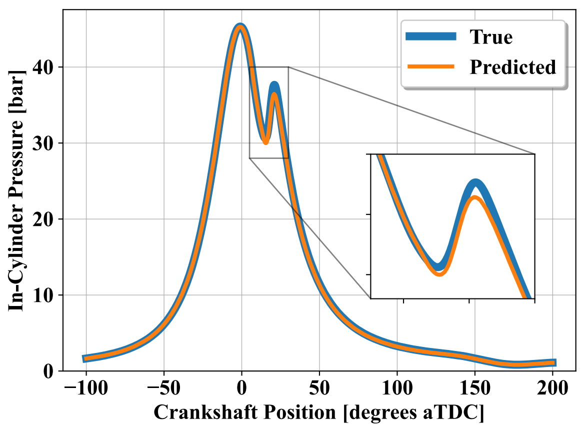
### 3.2 Model Performance Results

Once the optimal parameters for the model were found in the HPO, the model was trained again using a larger number of epochs. The time required to train the model for 100 epochs using the training data set with 46500 examples, the hyperparameter values found in the HPO process and the computational setup described in Sect. 2.2 was of 1028 seconds. For reference, using the same computational setup and the same training data set, the training time over 100 epochs of the model using the smallest model size from the HPO search space (2 hidden layers, 128 nodes each layer) was of 787 seconds, which yielded $R^2$ values of 0.99978, 0.99970, and 0.99970 on the training, validation, and test data sets, respectively. On the other hand, using the largest architecture from the HPO search space (10 hidden layers, 2048 nodes each layer) yielded a training time of 6440 seconds and $R^2$ scores of 0.99980, 0.99958, and 0.99940 on the training, validation, and test sets, respectively.

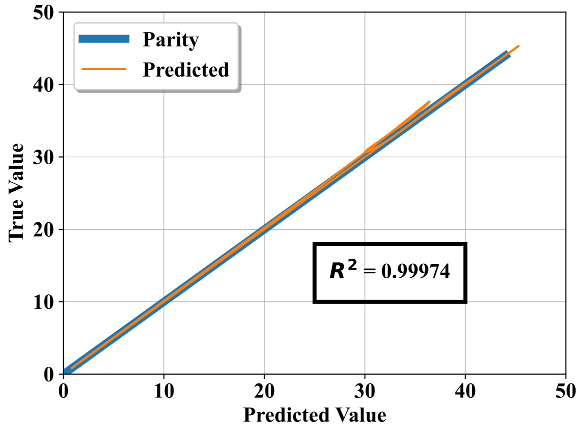
Figure 3 shows a double y-axis plot of the progression of the MSE loss value (left y-axis, plotted on log scale) as well as the $R^2$ score (right y-axis) for the training and validation sets during the training process. As it can be seen from Fig. 3, a plateau was reached for the MSE loss value at around epoch number 80, while the $R^2$ score reached a plateau earlier. It can most importantly be seen in this figure that the values of the MSE loss and $R^2$ score are very similar for the training and validation sets, indicating the model does not appear to be overfitting to the training data and generalizes well to unseen examples. A summary of the performance metrics for train, validation, and test sets is shown in Table 4.

Because each output of the model is a sequence of values that must be displayed together instead of a single value, it is difficult to provide visual representation of a large number of generated outputs of the model. For this reason, only three predictions from the test set are shown, along with the performance metrics calculated from them, to serve as examples and visually demonstrate how the numbers reported in Table 4 translate to pressure predictions. Two plots were generated for each example, one that displays the predicted and true pressure values as functions of crankshaft position, and one that shows the linearity of the relationship between true and predicted pressure values.

Figure 4 shows the results of a prediction that has an average MSE loss and $R^2$ score, in this case $4.4012 \cdot 10^{-2}$ and 0.99974, respectively. This cycle was chosen to be the representative for predictions with MSE and $R^2$ scores close to the average values calculated for the validation and test data sets (Table 4). It can be seen from Fig. 4a that there is a small discrepancy between the timing of the start of the pressure rise rate from combustion, in this particular case the combustion in the predicted pressure trace begins later. This also leads to a slightly lower peak pressure in the predicted pressure trace. However, this is a very mild discrepancy, and one that would be expected considering cycle-to-cycle variations in a real engine. This discrepancy is reflected in Fig. 4b where the predicted points, plotted in orange, are offset in a small section from the $y = x$ parity line, which is plotted in blue. In this linearity plot, any deviation from the blue line represents a mismatch between true and predicted value. Although it is very visible that there is a range of pressure values

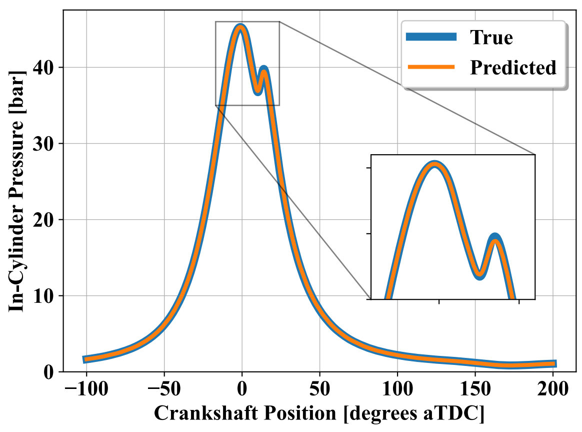


(a) Pressure Trace with Average Correlation

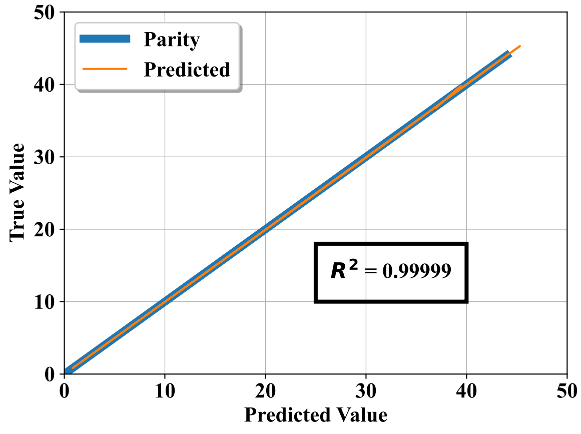


(b) Linear Fit of Pressure Prediction with Average Correlation

FIGURE 4: PREDICTION WITH AVERAGE CORRELATION

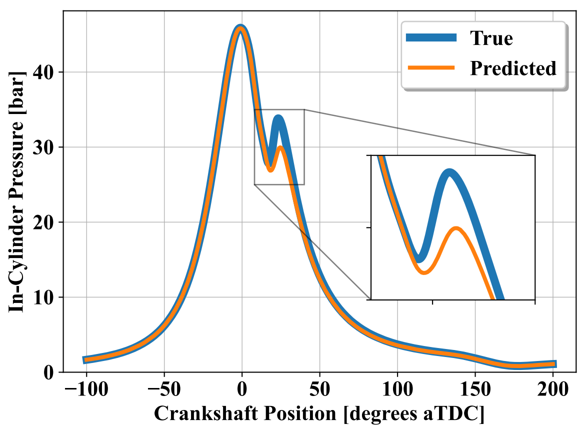


(a) Pressure Trace with Good Correlation

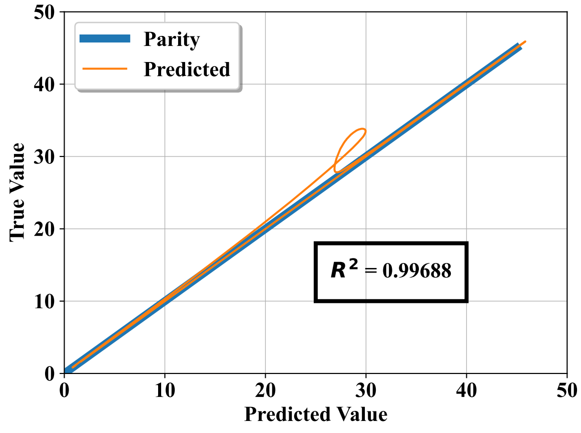


(b) Linear Fit of Pressure Prediction with Good Correlation

FIGURE 5: PREDICTION WITH GOOD CORRELATION



(a) Pressure Trace with Poor Correlation



(b) Linear Fit of Pressure Prediction with Poor Correlation

FIGURE 6: PREDICTION WITH POOR CORRELATION

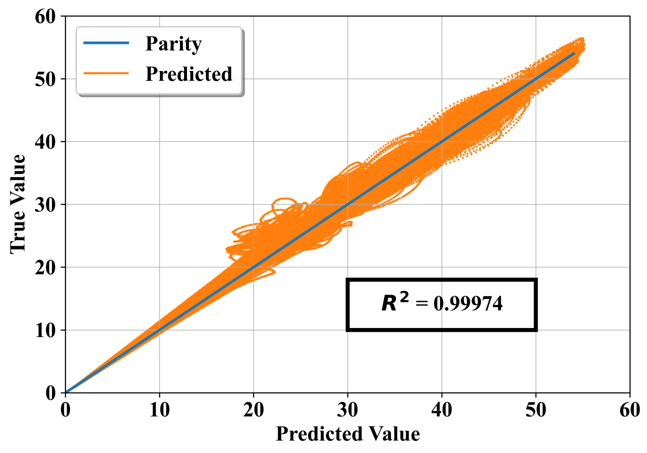


FIGURE 7: LINEAR FIT OF TEST SET

for which the prediction did not match the true label, the vast majority of points still lie along the $y = x$ line thus yielding an $R^2$ score of 0.99974.

In some cases the model was able to predict the pressure trace nearly perfectly, which is the case shown in Fig. 5. The MSE score for this example was evaluated to be $1.8804 \cdot 10^{-3}$, while the $R^2$ score was calculate to be 0.99999. As can be observed in Fig. 5a, all the elements of the pressure trace, including start of combustion and peak pressure, are captured in this example of the test set. This is further confirmed in Fig. 5b which shows that the blue parity line and the predicted values shown in orange are in near perfect agreement.

The last example portrays the other extreme in Fig. 6, a prediction with quite significant differences in the combustion portion of the pressure trace. The MSE loss for this example was of $7.6038 \cdot 10^{-1}$ and the $R^2$ score was of 0.99688. By analyzing Fig. 6a it is clear that the start of combustion time is delayed in the predicted value, which in turn yielded a decrease in the maximum pressure of combustion compared to the true value. The behavior is further confirmed in Fig. 6b where a long portion of the predicted values (orange) fall outside of the $y = x$ line (blue). The $R^2$ score for this prediction is lower for this example, however it still shows high linearity because it is able to capture the overall trend of the cycle.

Lastly, Fig. 7 shows the linear fit of all examples from the test set. Given how discrepancies between predicted and true values manifest in the linear fit plot as shown in Figures 4b, 5b, and 6b, it is expected to have a wide spread of points that deviate from the parity line if the number of examples in the same plot is large. Nonetheless, the linearity of the predictions is clear, which explains the high coefficient of determination of 0.99974.

#### 3.2.1 Model Performance Across Input Space.

It was noticed that most of the predictions in which the model demonstrated relatively poor performance were for cases near the edge of the exploration matrix of the input features. An example is shown in Fig. 6a where the combustion occurred very late in the cycle which indicates it is a cycle with late fuel injection. This is expected given that, while ML models in general can be very good at interpolating complex data, their accuracy significantly decreases when extrapolating to inputs that are outside the range of data the model was trained on [42]. However, it was found that there is one more factor that strongly influenced the performance of the model: the COV of the $IMEP_n$, referred to simply as COV from hereon out.

To illustrate this finding, 2D scatter plots that show each input feature plotted against each other were generated and are shown below. The color map in these figures represent the mean squared error of the validation and test sets, and the coefficient of variation of the points in the training set in Figs. 8 and 9, respectively. It can be seen by comparing Figs. 8 and 9 that in areas where the COV of the points used in the training set is high, the model tends to demonstrate high MSE. This can be reasoned by considering that a high COV means that for the same value of the inputs (within experimental uncertainty) the behavior of the output is less consistent. In the case of this compression ignition engine, as seen in Fig. 9, this tends to happen in regions of late injection timing for a given equivalence ratio, and also when the injection pressure is low, at approximately 450 bar in this case.

In addition to operating points where the COV is high, some seemingly random outliers are found which have high MSE where the COV is low. The authors hypothesise that this behavior is not random, and is an artifact of the location of the point within the experimental matrix, meaning that these outliers lie on the edge of the range of values of at least one input feature. To make this evident, color-coded and labeled boxes that mark the same set of points in all subplots are included in the figures. The points with a green box with label A, for example, lie at the edge of values of all three features. Point C in blue, lies at the edge of the equivalence ratio range for that injection pressure and injection timing. The author's explanation for these is based on the following. Considering that a data point can only be a part of either test, validation or training sets, if an edge case is present in the test/validation sets, then it was not used during the training process. Ultimately this means that, in those cases, the model is extrapolating, thus explaining the drop in performance.

While nothing can be done regarding these outliers since standard practice dictates the splitting of data between the different sets should be based on random sampling, reducing the COV of the data set is expected to improve the overall performance of the model. A COV below approximately 5% does appear to be low enough to maintain model performance based on the results shown in Figs. 8 and 9.

## 4. CONCLUSIONS AND FUTURE WORK

The findings of this work can be summarized by the following statements.

- Deeper networks appear to have better generalization characteristics compared to shallow networks in this application as demonstrated in the hyperparameter optimization results.
- The poor predictions of the model tend to be for examples near the edge of the experimental matrix of the input features and in operating points with high coefficient of variation.
- MLP models can be used to accurately predict in-cylinder pressure in compression ignition engines within the range

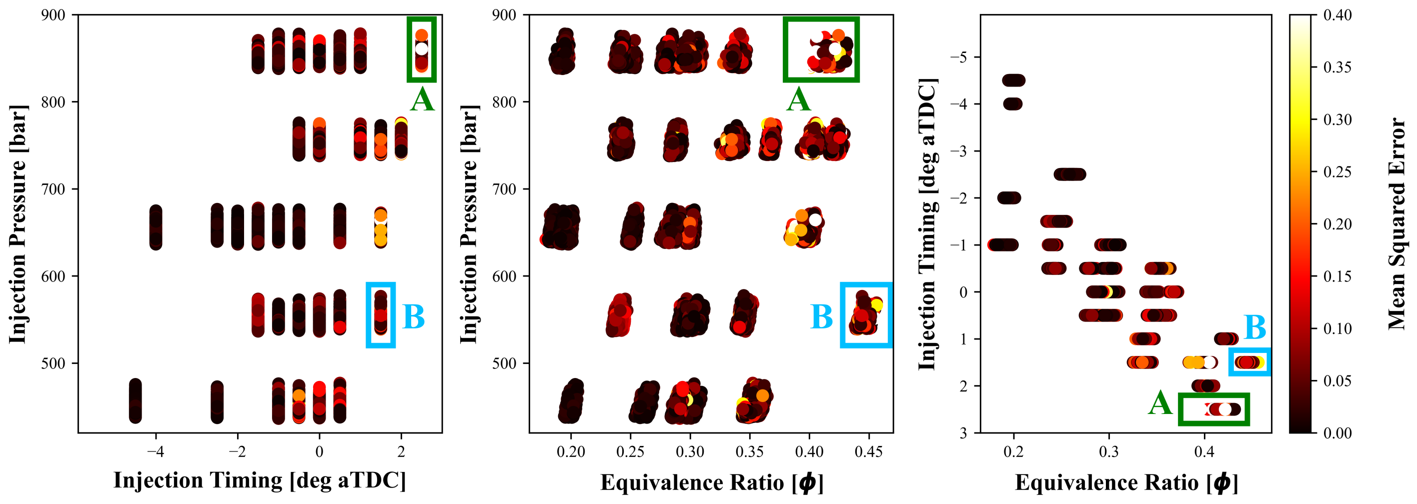


FIGURE 8: 2D SCATTER PLOT REPRESENTATION OF 3D INPUT FEATURE SPACE WITH COMMON MSE COLOR MAP FOR VALIDATION AND TEST SETS

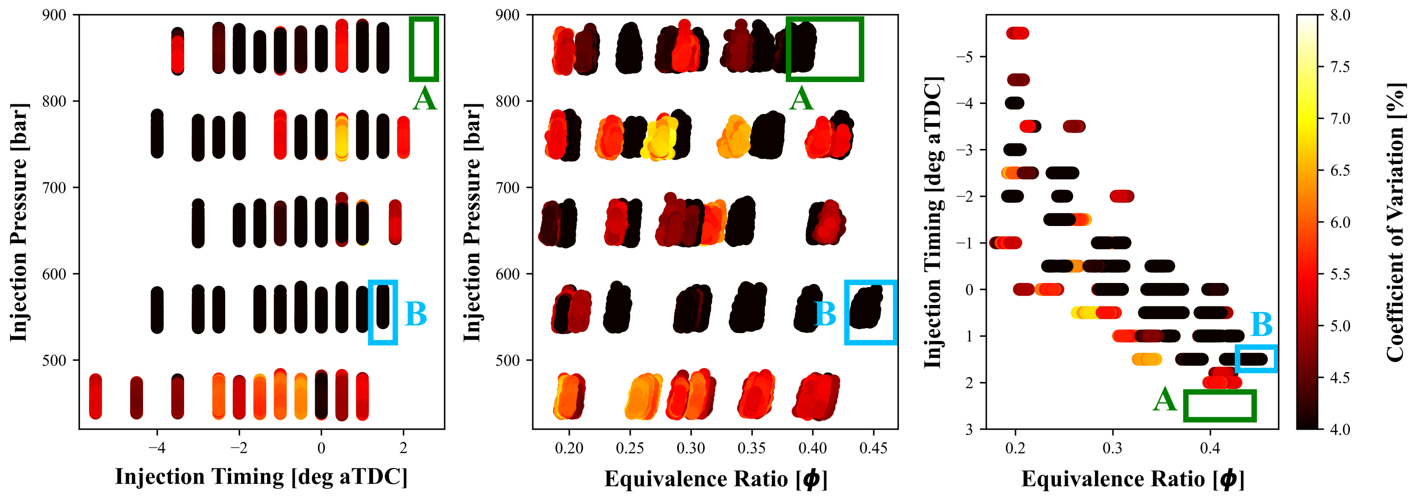


FIGURE 9: 2D SCATTER PLOT REPRESENTATION OF 3D INPUT FEATURE SPACE WITH COMMON COV COLOR MAP FOR TRAINING SET

of input data used for training using equivalence ratio, fuel injection pressure, and fuel injection timing as input features.

- The model described in this work demonstrates potential to be a promising tool for the modeling and optimization of internal combustion engines.

Interesting future work in this area can be done to investigate how the error in predictions of the cylinder pressure values propagates to other derived performance parameters such as $IMEP_n$, heat release rate, and efficiencies.

## ACKNOWLEDGMENTS

This work was made possible through funding from the U.S. Office of Naval Research under Award No. N00014-22-1-2001 and through the donation of liquid fuels used in experimental testing by Haltermann Carless. The authors would like to also acknowledge Dr. Zhongnan Ran for sharing his experimental testing knowledge on the single-cylinder facility. In addition, the authors appreciate the insightful conversations and guidance of Prof. Yifan Sun from the Dept. of Computer Science and Prof. Nilanjan Chakraborty from the Department of Mechanical Engineering, both at Stony Brook University.

## REFERENCES

[1] Giglio, Veniero and di Gaeta, Alessandro. "Novel regression models for wiebe parameters aimed at 0D combustion simulation in spark ignition engines." *Energy* Vol. 210 (2020): p. 118442. DOI 10.1016/J.ENERGY.2020.118442.

[2] Yang, Ruinan, Ran, Zhongnan, Ristow Hadlich, Rodrigo and Assanis, Dimitris. "A Double-Wiebe Function for Reactivity Controlled Compression Ignition Combustion Using Reformate Diesel." *Journal of Energy Resources Technology* Vol. 144 No. 11 (2022): p. 112301.

[3] Yang, Ruinan, Ran, Zhongnan, Ristow Hadlich, Rodrigo

and Assanis, Dimitris. “A Double-Wiebe Function for Reactivity Controlled Compression Ignition Combustion Using Reformate Diesel.” *Journal of Energy Resources Technology* Vol. 144 No. 11 (2022): p. 112301.

[4] Cerri, Tarcisio, Onorati, Angelo and Mattarelli, Enrico. “1D engine simulation of a small HSDI diesel engine applying a predictive combustion model.” *Journal of Engineering for Gas Turbines and Power* Vol. 130 (2008). DOI 10.1115/1.2747258/470382.

[5] Assanis, Dimitris, Engineer, Nayan, Neuman, Paul and Wooldridge, Margaret. “Computational Development of a Dual Pre-Chamber Engine Concept for Lean Burn Combustion.” 2016. DOI 10.4271/2016-01-2242.

[6] Nikiforakis, I, Guleria, G, Koraiem, M and Assanis, D. “Understanding Pre-Chamber Combustion Performance in a Closed-Cycle Model of a Novel Rotary Engine.” *SAE Technical Paper* (2022)DOI 10.4271/2022-01-0396.

[7] Guleria, Gaurav, Lopez-Pintor, Dario, Dec, John E. and Assanis, Dimitris. “A comparative study of gasoline skeletal mechanisms under partial fuel stratification conditions using large eddy simulations.” *International Journal of Engine Research* Vol. 23 (2022): pp. 1658–1677. DOI 10.1177/14680874211031370.

[8] Guleria, Gaurav, Lopez-Pintor, Dario, Dec, John E and Assanis, Dimitris. “Development and evaluation of a skeletal mechanism for EHN additized gasoline mixtures in large Eddy simulations of HCCI combustion.” *International Journal of Engine Research* (2023): p. 14680874231178099.

[9] Tang, Yuanyuan, Zhang, Jundong, Gan, Huibing, Jia, Baozhu and Xia, Yu. “Development of a real-time two-stroke marine diesel engine model with in-cylinder pressure prediction capability.” *Applied Energy* Vol. 194 (2017): pp. 55–70. DOI 10.1016/J.APENERGY.2017.03.015.

[10] Nuss, Eugen, Ritter, Dennis, Wick, Maximilian, Andert, Jakob, Abel, Dirk and Albin, Thivaharan. “Reduced order modeling for multi-scale control of low temperature combustion engines.” *Notes on Numerical Fluid Mechanics and Multidisciplinary Design* Vol. 141 (2019): pp. 167–181. DOI 10.1007/978-3-319-98177-2_11/FIGURES/5.

[11] Mayhew, Christopher G., Knierim, Karl Lukas, Chaturvedi, Nalin A., Park, Sungbae, Ahmed, Jasim and Kojic, Aleksandar. “Reduced-order modeling for studying and controlling misfire in four-stroke HCCI engines.” *Proceedings of the IEEE Conference on Decision and Control* (2009): pp. 5194–5199DOI 10.1109/CDC.2009.5400597.

[12] Solmaz, Ozgur, Gurbuz, Habib and Karacor, Mevlut. “Comparison of artificial neural network and fuzzy logic approaches for the prediction of in-cylinder pressure in a spark ignition engine.” *Journal of Dynamic Systems, Measurement and Control, Transactions of the ASME* Vol. 142 (2020). DOI 10.1115/1.4047014/1082935.

[13] Aliramezani, Masoud, Koch, Charles Robert and Shahbakhti, Mahdi. “Modeling, diagnostics, optimization, and control of internal combustion engines via modern machine learning techniques: A review and future directions.” *Progress in Energy and Combustion Science* Vol. 88 (2022). DOI 10.1016/j.pecs.2021.100967.

[14] Zhou, Lei, Song, Yuntong, Ji, Weiqi and Wei, Haiqiao. “Machine learning for combustion.” *Energy and AI* Vol. 7 (2022): p. 100128. DOI 10.1016/J.EGYAI.2021.100128.

[15] Almeldein, Ahmed and Dam, Noah Van. “Accelerating Chemical Kinetics Calculations With Physics Informed Neural Networks.” *Proceedings of ASME 2022 ICE Forward Conference, ICEF 2022* (2022)DOI 10.1115/ICEF2022-90371.

[16] Liu, Jinlong, Huang, Qiao, Ulishney, Christopher and Dumitrescu, Cosmin E. “Machine learning assisted prediction of exhaust gas temperature of a heavy-duty natural gas spark ignition engine.” *Applied Energy* Vol. 300 (2021): p. 117413. DOI 10.1016/J.APENERGY.2021.117413.

[17] Yusaf, Talal F., Buttsworth, D. R., Saleh, Khalid H. and Yousif, B. F. “CNG-diesel engine performance and exhaust emission analysis with the aid of artificial neural network.” *Applied Energy* Vol. 87 (2010): pp. 1661–1669. DOI 10.1016/J.APENERGY.2009.10.009.

[18] Togun, Necla Kara and Baysec, Sedat. “Prediction of torque and specific fuel consumption of a gasoline engine by using artificial neural networks.” *Applied Energy* Vol. 87 (2010): pp. 349–355. DOI 10.1016/J.APENERGY.2009.08.016.

[19] Khatri, Kamal Kishore, Singh, Mandeep and Khatri, Narendra. “An artificial neural network model for the prediction of performance and emission parameters of a CI engine-operated micro-tri-generation system fueled with diesel, Karanja oil, and Karanja biodiesel.” *Fuel* Vol. 334 (2023): p. 126549. DOI 10.1016/J.FUEL.2022.126549.

[20] Pillai, Rinav, Triantopoulos, Vassilis, Berahas, Albert S., Brusstar, Matthew, Sun, Ruonan, Nevius, Tim and Boehman, André L. “Modeling and Predicting Heavy-Duty Vehicle Engine-Out and Tailpipe Nitrogen Oxide (NO x) Emissions Using Deep Learning.” *Frontiers in Mechanical Engineering* Vol. 8 (2022): p. 11. DOI 10.3389/FMECH.2022.840310/BIBTEX.

[21] Bai, Femilda Josephin Joseph Shobana, Shanmugaiah, Kaliraj, Sonthalia, Ankit, Devarajan, Yuvarajan and Varuvel, Edwin Geo. “Application of machine learning algorithms for predicting the engine characteristics of a wheat germ oil–Hydrogen fuelled dual fuel engine.” *International Journal of Hydrogen Energy* (2022)DOI 10.1016/j.ijhydene.2022.11.101.

[22] Vaughan, Adam and Bohac, Stanislav V. “Real-time, adaptive machine learning for non-stationary, near chaotic gasoline engine combustion time series.” *Neural Networks* Vol. 70 (2015): pp. 18–26. DOI 10.1016/J.NEUNET.2015.04.007.

[23] Shamshirband, Shahaboddin, Tabatabaei, Meisam, Aghbashlo, Mortaza, Yee, Por Lip and Petković, Dalibor. “Support vector machine-based exergetic modelling of a DI diesel engine running on biodiesel–diesel blends containing expanded polystyrene.” *Applied Thermal Engineering* Vol. 94 (2016): pp. 727–747. DOI 10.1016/J.APPLTHERMALENG.2015.10.140.

[24] Mariani, Viviana Cocco, Och, Stephan Hennings, dos Santos Coelho, Leandro and Domingues, Eric. "Pressure prediction of a spark ignition single cylinder engine using optimized extreme learning machine models." *Applied Energy* Vol. 249 (2019): pp. 204–221. DOI 10.1016/J.APENERGY.2019.04.126.

[25] Badra, Jihad A., Khaled, Fethi, Tang, Meng, Pei, Yuanjiang, Kodavasal, Janardhan, Pal, Pinaki, Owoyele, Opeoluwa, Fuetterer, Carsten, Mattia, Brenner and Aamir, Farooq. "Engine Combustion System Optimization Using Computational Fluid Dynamics and Machine Learning: A Methodological Approach." *Journal of Energy Resources Technology, Transactions of the ASME* Vol. 143 (2021). DOI 10.1115/1.4047978/1086007.

[26] Mohan, Balaji and Badra, Jihad. "An automated machine learning framework for piston engine optimization." *Applications in Energy and Combustion Science* Vol. 13 (2023): p. 100106. DOI 10.1016/J.JAECS.2022.100106.

[27] Badra, Jihad, Khaled, Fethi, Sim, Jaeheon, Pei, Yuanjiang, Viollet, Yoann, Pal, Pinaki, Futterer, Carsten, Brenner, Mattia, Som, Sibendu, Farooq, Aamir and Chang, Junseok. "Combustion System Optimization of a Light-Duty GCI Engine Using CFD and Machine Learning." *SAE Technical Papers* Vol. 2020-April (2020). DOI 10.4271/2020-01-1313.

[28] Mohan, Balaji, Tang, Meng, Badra, Jihad, Pei, Yuanjiang and Traver, Michael. "Machine Learning and Response Surface-Based Numerical Optimization of the Combustion System for a Heavy-Duty Gasoline Compression Ignition Engine." *SAE Technical Papers* (2021)DOI 10.4271/2021-01-0190.

[29] Owoyele, Opeoluwa and Pal, Pinaki. "A novel machine learning-based optimization algorithm (ActivO) for accelerating simulation-driven engine design." *Applied Energy* Vol. 285 (2021): p. 116455. DOI 10.1016/J.APENERGY.2021.116455.

[30] Posch, S., Winter, H., Zelenka, J., Pirker, G. and Wimmer, A. "Development of a tool for the preliminary design of large engine prechambers using machine learning approaches." *Applied Thermal Engineering* Vol. 191 (2021): p. 116774. DOI 10.1016/J.APPLTHERMALENG.2021.116774.

[31] Gharehghani, Ayat, Abbasi, Hamid Reza and Alizadeh, Pouria. "Application of machine learning tools for constrained multi-objective optimization of an HCCI engine." *Energy* Vol. 233 (2021): p. 121106. DOI 10.1016/J.ENERGY.2021.121106.

[32] Harris, Charles R., Millman, K. Jarrod, van der Walt, Stéfan J., Gommers, Ralf, Virtanen, Pauli, Cournapeau, David, Wieser, Eric, Taylor, Julian, Berg, Sebastian, Smith, Nathaniel J., Kern, Robert, Picus, Matti, Hoyer, Stephan, van Kerkwijk, Marten H., Brett, Matthew, Haldane, Allan, del Río, Jaime Fernández, Wiebe, Mark, Peterson, Pearu, Gérard-Marchant, Pierre, Sheppard, Kevin, Reddy, Tyler, Weckesser, Warren, Abbasi, Hameer, Gohlke, Christoph and Oliphant, Travis E. "Array programming with NumPy." *Nature* Vol. 585 (2020): pp. 357–362. DOI 10.1038/S41586-020-2649-2.

[33] Paszke, Adam, Gross, Sam, Massa, Francisco, Lerer, Adam, Google, James Bradbury, Chanan, Gregory, Killeen, Trevor, Lin, Zeming, Gimelshein, Natalia, Antiga, Luca, Desmaison, Alban, Xamla, Andreas Köpf, Yang, Edward, Devito, Zach, Nabla, Martin Raison, Tejani, Alykhan, Chilamkurthy, Sasank, Ai, Qure, Steiner, Benoit, Facebook, Lu Fang, Facebook, Junjie Bai and Chintala, Soumith. "PyTorch: An Imperative Style, High-Performance Deep Learning Library.": pp. 8024–8035. 2019. Curran Associates, Inc.

[34] Akiba, Takuya, Sano, Shotaro, Yanase, Toshihiko, Ohta, Takeru and Koyama, Masanori. "Optuna: A Next-generation Hyperparameter Optimization Framework." *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2019): pp. 2623–2631DOI 10.1145/3292500.3330701.

[35] Rumelhart, David E., Hinton, Geoffrey E. and Williams, Ronald J. "Learning representations by back-propagating errors." *Nature 1986 323:6088* Vol. 323 (1986): pp. 533–536. DOI 10.1038/323533a0.

[36] Kingma, Diederik P. and Ba, Jimmy Lei. "Adam: A Method for Stochastic Optimization." *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2014)DOI 10.48550/arxiv.1412.6980.

[37] Nair, Vinod and Hinton, Geoffrey E. "Rectified Linear Units Improve Restricted Boltzmann Machines.": pp. 807–814. 2010.

[38] Maas, Andrew L, Hannun, Awni Y and Ng, Andrew Y. "Rectifier Nonlinearities Improve Neural Network Acoustic Models." *Proceedings of the international conference on machine learning* Vol. 30 (2013).

[39] Bergstra, James, Bardenet, Rémi, Bengio, Yoshua and Kégl, Balázs. "Algorithms for Hyper-Parameter Optimization." *Advances in Neural Information Processing Systems* Vol. 24 (2011).

[40] Mhaskar, Hrushikesh, Liao, Qianli and Poggio, Tomaso. "When and Why Are Deep Networks Better Than Shallow Ones?" *Proceedings of the AAAI Conference on Artificial Intelligence* Vol. 31 (2017): pp. 2343–2349. DOI 10.1609/AAAI.V31I1.10913.

[41] Srivastava, Nitish, Hinton, Geoffrey, Krizhevsky, Alex and Salakhutdinov, Ruslan. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." *Journal of Machine Learning Research* Vol. 15 (2014): pp. 1929–1958.

[42] McCartney, Michael, Haeringer, Matthias and Polifke, Wolfgang. "Comparison of Machine Learning Algorithms in the Interpolation and Extrapolation of Flame Describing Functions." *Journal of Engineering for Gas Turbines and Power* Vol. 142 (2020). DOI 10.1115/1.4045516/1069492.