MATERIALS AND MANUFACTURING PROCESSES

Evolutionary Neural Architecture Search on Transformers for RUL Prediction

Hyunho Mo, Giovanni Iacca

University of Trento, Via Sommarive 9, 38123 Trento, Italy
hyunho.mo@unitn.it, giovanni.iacca@unitn.it

ARTICLE HISTORY
Compiled February 21, 2023

ABSTRACT
Remaining useful life (RUL) predictions are a key enabler for achieving efficient maintenance in the context of Industry 4.0. Data-driven approaches, in particular employing deep neural networks (DNNs), have shown success in the RUL prediction task. However, although their architecture considerably affects performance, DNNs are usually handcrafted by human experts via a labor-intensive design process. To overcome this issue, we propose a neural architecture search (NAS) technique that explores a search space using a genetic algorithm (GA). It automatically discovers the optimal architectures of Transformers for RUL predictions. Our GA allows an efficient search, by making use of a performance predictor, updated at every generation, that reduces the needed network training. To our knowledge, this is the first work to optimize the architecture of Transformers for RUL predictions using evolutionary computation. We evaluate the performance of the found solutions on a widely-used benchmark dataset, the CMAPSS, based on RMSE and *s*-score. In comparison with the state-of-the-art, the Transformer obtained by our NAS method outperforms other recent handcrafted DNNs in terms of RMSE, and is comparable regarding *s*-score. Our results demonstrate that the proposed method provides better prediction accuracy with less human effort compared to other data-driven approaches.

KEYWORDS
Predictive, Maintenance, Evolutionary, Algorithm, Neural, Architecture, Search, Transformers

1. Introduction

Predictive maintenance (PdM) is an enabling technology that enhances the reliability of critical systems and cuts down maintenance costs. It aims to develop an effective maintenance policy based on predictions of the state of the system at hand. Accurate remaining useful life (RUL) predictions are a key component to PdM, considering that precisely predicted RUL enables making better maintenance decisions[1], allowing to perform timely maintenance before failures occur. In fact, providing reliable RUL predictions can allow users (e.g., plant owners or managers) to save expensive maintenance procedures, which often include costly inspection and imposed stops to the operation of the system. On the other hand, being able to anticipate, within a reasonable time window (i.e., neither too early nor too late) any potential system fault not only can enhance the quality of manufacturing processes and their output products, but also cut losses caused by any unexpected downtime, even without regular and excessive maintenance to prevent failures. Providing accurate RUL predictions is, however, a difficult task. In fact, data about faults are usually scarce (because these events typically occur less frequently than normal operational conditions). Moreover, even when monitoring data about the system are available, these may not contain enough information to make the predictions (e.g., some states of the system that would be needed to predict faults may not be observable due to technical limitations or other kinds of practical constraints, such as lack of sensors).

Driven by the above motivations, many research works focused on RUL prediction have been carried out in the past decade. Since as mentioned data related to faults are usually scarce, these works have mainly focused on developing successful approaches that can make precise RUL predictions with a limited amount of data. However, recent advances in machine learning (ML) have helped overcome this limitation. In fact, successful ML models can predict RUL accurately by learning patterns on scarce data, even without inquiring into the underlying physics of the monitored process. Thus, data-driven approaches based on ML have become popular [2].

Particularly, artificial neural networks (ANNs) have been employed to develop data-driven approaches for RUL predictions, and their architectures have become more and more complex. More specifically, multi-layer perceptron (MLPs) and convolutional neural networks (CNNs) [3], both instances of traditional back-propagation neural networks (BPNNs), have been the earliest architectures applied to the RUL prediction task. Later, RUL has been predicted by recurrent neural networks (RNNs), and they have been also used in combination with CNNs [4,5]. More recently, autoencoder (AE) based models have been utilized to analyze temporal patterns on time series data [6,7], and an attention mechanism on top of the DL architecture [8] has shown a large success in the RUL prediction task.

While the above DNNs have shown promising performance in RUL predictions, those need a significant amount of human effort to be developed and used for a specific RUL task. This is because the performance of those networks largely depends on their architecture, which is difficult to design even for experts who are specialized in DL and experienced in PdM. Such a design process involves finding appropriate values for the parameters related to the architecture, and these values are usually ruled by empirical evidences, or come out from trial-and-error. This may not be an efficient way to develop a RUL prediction tool that fully exploits the capability of DL models.

To address the aforementioned limitations, in the present paper we apply neural architecture search (NAS), that is a technique for automating the DNN design process. Generally, different optimization techniques can be used to realize NAS. Among them, we implement a genetic algorithm (GA) that incorporates a surrogate model into the evolutionary computation. In this work, the type of DNN to be designed is the Transformer [9], one of the current best-performing DL models, whose main feature is that it can draw long-term dependencies based exclusively on attention mechanisms. Albeit powerful, the performance of the Transformer is affected by its architecture parameters, and these parameters define a combinatorial search space. Here, we propose to use evolutionary computation to find the optimal architecture(s) by exploring this search space effectively.

This search can be, however, computationally expensive. In fact, considering that the training of a single Transformer architecture is *per se* computationally expensive, we reduce the computational complexity by employing, in the evaluation process conducted within the GA, a surrogate model that eschews constructing and training every single architecture appeared during the search. To this aim, we use a probabilistic regression model via gradient boosting, namely natural gradient boosting[10], as model performance predictor. This predictor is initialized with a predefined number of evenly distributed samples in the search space, and updated across generations by retraining it with only a few solutions that are expected to have good fitness. By doing so, we can improve the quality of the solutions found by our evolutionary search.

To evaluate the quality of the found solutions, we consider the *de facto* standard benchmark for RUL predictions, i.e., the commercial modular aero-propulsion system simulation (CMAPSS) dataset[11] by NASA. On each of its four sub-datasets, we search for the optimal Transformer architecture and compare it to the different methods in the existing literature, in terms of performance.

In summary, our key contributions are the following:

- We introduce a GA which integrates efficient retraining of a performance predictor in its evolutionary process.
- Our algorithm successfully finds better solutions than a space-filling sampling method.
- In 3 out of 4 sub-datasets, we obtain solutions that outperform the state-of-the-art methods w.r.t. the prediction error.
- To our knowledge, our work is the first study that proposes to optimize the architecture of the Transformer to be used for RUL predictions by means of evolutionary computation.

The rest of the paper is structured as follows: Section 2 explains the background concepts on the Transformer used for RUL predictions. Then, we specify the architecture parameters to be optimized and introduce the proposed GA based NAS technique. In Section 3, the details of our experimentation and the experimental results are presented. The following section, Section 4, discusses the conclusions of this work.

2. Materials and methods

This section details our proposed method. In Section 2.1, we introduce the backbone network that is optimized in the NAS process. To apply the evolutionary optimization, the architecture parameters are encoded as "individuals", following the description given in Section 2.2. The resulting search space, defined by these parameters and their bounds, is then explored by the GA. Moreover, the performance prediction mechanism introduced in Section 2.3 is considered to reduce the computational resources needed to evaluate all the individuals during the search. Finally, Section 2.4 presents the proposed algorithm which allows the interaction between the GA and the predictor during the evolutionary search.

2.1. Transformers

Transformers[9] model long-term dependencies relying entirely on an attention mechanism, i.e., without employing recurrence and convolutions. They have shown success in many ML applications: computer vision (CV)[12,13], speech recognition[14,15], and natural language processing (NLP)[16,17]. Considering that Transformers can handle long-term dependencies in sequential data while dismissing either RNNs or CNNs, they have been used for a variety of tasks dealing with time series data; Li et al.[18] introduced a Transformer network for time series forecasting and observed that its performance on real-world datasets was comparable to that of recent RNN based methods.

The anomaly Transformer, i.e., a Transformer variant with an anomaly-attention mechanism that is able to identify outlier points in time series, was developed in[19]. In[20], it was shown that a Transformer based framework can be used for unsupervised representation learning of time series, providing promising results on benchmark datasets for time series regression and classification[20].

Following these previous works, here we hypothesize that Transformers can be effectively used for the RUL prediction task by analyzing multi-sensor time series data. These sensor readings are typically sequential data, and as such RNNs have been widely used to analyze temporal patterns that appear in the sequences[5,21]. However, the training of such RNNs is computationally expensive compared to feed-forward networks (FFNs). Moreover, RNNs are vulnerable to gradient vanishing and explosion, while Transformers do not suffer from such problems because their structure does not rely on RNN modules and depends only on self-attention mechanisms. Although Transformers have been widely used in many applications as discussed above, most of the Transformer networks proposed are handcrafted. Typically, their layout design is determined empirically, and the relation between the architecture of Transformers and their performance has not been thoroughly discussed. In contrast, in our work the architectures of Transformers are automatically designed in the direction of improving RUL prediction performance. More specifically, we adopt the Transformer network introduced in[22] as our backbone network, whose architecture is optimized by our proposed algorithm.

In the following, we formulate the RUL prediction task and describe the background concepts of the Transformer network.

2.1.1. Problem formulation

The present work deals with a purely data-driven RUL prediction approach that does not need a priori knowledge regarding the physics underlying the degradation of target components. Figure 1 visualizes the flow of a data-driven RUL prediction task, highlighting that it is addressed by using a black-box model. According to this scheme, the target component has installed multiple sensors that measure some physical properties which are relevant to RUL predictions. These monitoring data consist of multivariate time series and are processed by a sliding window to prepare fixed-length sequential data. Then, a black-box model outputs a RUL value from its input. In this work, the black-box model is the Transformer network, which is trained by minimizing the training loss. We derive the loss function from the discrepancy between the predicted and the actual RUL (on training data with ground truth). During inference after training, the trained Transformer network produces the predicted RUL for the given input sequence data. Eventually, this prediction can contribute to making optimal decisions for maintenance management.

We can formulate the task described above, as follows. Let T and S indicate the length of a time window and the number of sensors respectively. We consider an input sequence data X determined by the window sliding over the multi-sensor measurements. In other words, the input sequence comprises the measurements of the S sensors over T timesteps, and each position in the sequence corresponds to each timestep. This sequence can be written as $X = [x_1, \dots, x_T]^\top$ with $x_t \in \mathbb{R}^{\mathrm{S}}$.

Considering a mini-batch of bs training samples, the loss function $l(\cdot)$ of the network corresponds to the Mean Squared Error (MSE) between the output of the network,

$Y = [y_1, \ldots, y_{bs}]$ and its ground truth labels, $Z = [z_1, \ldots, z_{bs}]$:

$$l(Y, Z) = \frac{1}{bs} \sum_{j=1}^{bs} (y_j - z_j)^2 \tag{1}$$

where y_j denotes the predicted RUL w.r.t. X_j while z_j represents the ground truth RUL for the j-th sample X_j. Note that in this work we always take the RUL value at the last timestep T of the sequence X.

2.1.2. Embeddings and positional encoding

From the input sequence, we need to obtain vectors that embed the sensor measurements at each timestep. In the Transformer, an input embedding layer converts the input sensor measurements to vectors of dimension d_{model}. While RNNs process one position at a time in the sequence, the Transformer can accept all the positions at once. To discriminate the relative or absolute position of the measurement, a positional encoding step follows the input embedding. When it comes to the RUL prediction model, the injection of the positional information is crucial, because it allows capturing the degradation pattern that appears in the input sequence made of sensor measurements over time. To capture the ordering of measurements, we use sinusoidal functions, which can produce distinct vectors depending on the position:

$$\begin{aligned} PE_{(t,2i)} &= \sin\left(t/10000^{2i/d_{model}}\right) \\ PE_{(t,2i+1)} &= \cos\left(t/10000^{2i/d_{model}}\right) \end{aligned} \tag{2}$$

where t indicates the position of the measurement, and i represents the entry of the vector. This way, unique integers from different positions correspond to different frequencies of the sinusoidal functions. The dimension of the obtained positional embeddings must be d_{model}, so that these can be added to the input embeddings (see the bottom of Figure 2).

2.1.3. Model architecture

The Transformer follows the encoder-decoder architecture drawn in Figure 2. Both the encoder and decoder are made up of a multi-head attention layer followed by a feed-forward layer. The attention layer contains one of the most widely adopted attention modules, i.e., the Scaled Dot-Product Attention module, shown in Figure 3 (right). This module performs a matrix multiplication first. The outcome of the dot-product is then scaled by a constant factor $1/d_{model}$[9]:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_{model}}}\right) V \tag{3}$$

where Q, K and V denote, respectively, a set of queries, keys and values. Equation (3) explains that the attention modules output a weighted sum of the values. Its weights are calculated as the softmax output w.r.t the dot-product which measures the similarity of each query to the keys.

Similar to the use of multiple feature maps in CNNs, Transformers take advantage of multi-head attention, by making use of h independent heads. To realize this, h pairs of different linear projections transform d_{model}-dimensional queries, keys, and values to d_k, d_k, and d_v dimensions, respectively. Each of those projected vectors proceeds to the designated head, as shown in Figure 3. The heads apply the attention function, which is defined in Equation (3), in parallel. The following concatenation layer concatenates the outputs of the h heads, and those are projected to d_{model} dimensions with the last linear projection layer:

$$\begin{gathered}\text{Multi-Head}(Q, K, V) = \text{Concat}\left(\text{head}_1, \ldots, \text{head}_h\right) W^O \\ \text{where head}_i = \text{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right)\end{gathered} \tag{4}$$

where the parameter matrices $W_i^Q \in \mathbb{R}^{\mathrm{d_{model}} \times \mathrm{d}_k}$, $W_i^K \in \mathbb{R}^{\mathrm{d_{model}} \times \mathrm{d}_k}$, and $W_i^V \in \mathbb{R}^{\mathrm{d_{model}} \times \mathrm{d}_v}$ are the linear projections preceding the attention, and $W^O \in \mathbb{R}^{h\mathrm{d}_v \times \mathrm{d_{model}}}$ denotes the parameter matrix for the projection of the concatenated output.

The output of the above attention proceeds to a FFN that comprises of two linear transformations:

$$\text{FFN}(X) = \max\left(0, XW_1 + B_1\right) W_2 + B_2. \tag{5}$$

The first layer linearly transforms from d_{model} to d_{ff} dimensions, and this inner layer uses the rectified linear unit (ReLU) as the activation function. Then, the following transformation produces the output of the FFN. The sub-layers explained above, i.e., the multi-head attention and the FFN, are the building blocks for both the encoder and decoder.

As depicted in Figure 2, the Transformer has a residual connection[23] for each sub-layer, and the outcome of this connection is followed by layer normalization[24]. Indicating with $SubLayer(\cdot)$ the function of the sub-layer, the output for input X of the sub-layer can be written as: $LayerNorm(X + SubLayer(X))$. The dimension of this sub-layer output is d_{model}, such that the sub-layer in the Transformer always receives a d_{model}-dimensional input and produces a d_{model}-dimensional output.

Different from vanilla Transformers[9], the Transformer network considered in this work contains two types of encoder: a sensor encoder, and a timestep encoder. The former is deployed to weigh different sensors by self-attention, while the latter serves to extract the feature from different timesteps. Each type of encoder is structured by stacking N_{enc} identical layers, and the two encoders work in parallel. The sensor encoder and the timestep encoder generate different representations, denoted by F_s and F_t respectively. The following feature fusion layer combines them as follows:

$$\text{Fusion}\left(F_s, F_t\right) = \text{Concat}\left(F_s, F_t\right) W^F. \tag{6}$$

In essence, this layer first concatenates the two representations, and then linearly projects the concatenation using the parameter matrix $W^F \in \mathbb{R}^{(\mathrm{d}_k + \mathrm{T}) \times \mathrm{d_{model}}}$.

The decoder in the Transformer network is in charge of predicting the RUL value at the end of the given X. The last point of the sequence corresponds to the current time at which the prediction is made. For predicting the current RUL, sensor readings at the end of the sequence are much more important compared to those at the beginning of the sequence, because traces of degradation on the sensor readings near the current timestep have a large impact on the RUL. In other words, the last α timesteps of the given T length sequence contain crucial information regarding the current RUL prediction. Taking this into consideration, the values for the last 4 timesteps of the encoder input X are adopted as the decoder input.

The decoder is constructed by piling up N_{dec} layers, each one composed of two attention blocks followed by one FNN. Similar to the encoder, the attention block at the beginning takes the embeddings w.r.t. the decoder input and computes their values. The next attention block receives the output of the previous multi-head attention block, as well as the output of the feature fusion layer, to look back at what the encoder input sequences were. Finally, the output layer at the very end of the Transformer converts the decoder output to the predicted RUL.

2.2. Individual encoding

Our base model described above involves many parameters that can vary the architecture of the Transformer network. Variations on the Transformer architecture can cause changes in performance[9], and their effect is typically investigated empirically, by changing one parameter at a time[9,25]. However, this is a labor-intensive design process, and the empirical evidence obtained in this way does not reflect the complex dependencies between the architecture parameters.

Different from the naïve approach, here we consider the Transformer architecture parameter choice as an optimization problem, aiming to discover the maximum possible performance by systematically diversifying all the architecture design parameters. To this end, we use evolutionary optimization. In this case, each "individual" in the population handled by the GA encodes a candidate solution for the Transformer network design problem, i.e., a vector representation of all the parameters that can vary the Transformer architecture.

Table 1 presents the 11 parameters that configure the Transformer architecture. The first six parameters are related to the dimensions of the representations; among these, the first three parameters determine the vector dimensions in one sub-layer, i.e., the multi-head attention, while the remaining three parameters determine the inner dimensions in another sub-layer, the FFN. To limit the possible combinations, each dimension parameter is divided by a fixed value of 4 w.r.t. the range of that parameter. The lower bound of the range is mainly based on d_{model}, since the smallest possible d_{model} should be larger than the upper bound of h, 16. The specific range of dimensions is determined empirically. In particular, based on preliminary observations, we have found that a d_{model} of 16 cannot decrease the training loss and for this reason d_{model} should be larger than 24. From this lower bound $6 = 24/4$, the upper bound is set to $25 = 100/4$, so that we can avoid combinatorial explosion by limiting the range of the parameter to 20 integer values. We consider the same range for all six parameters regarding the dimension.

The next three parameters indicate the number of attention heads. Its upper bound, 16, is inspired by the existing work[26], which examines the correlation between number of heads and performance.

Finally, the last two parameters indicate the number of identical layers stacked when we generate the encoder and decoder. The upper bound for the number of stacks is set to 3. With these upper bounds, the largest network fits the available memory used in our work.

Overall, the search space defined by the above 11 parameters includes $2.36 \times 10^{12} \approx 20^6 \cdot 16^3 \cdot 3^2$ Transformer configurations.

2.3. Fitness evaluation and performance predictor

NAS consists in solving an optimization problem that is formally written in:

$$a^* = \arg\min_a f(a) \tag{7}$$

where a denotes a given architecture from the search space, and a^* indicates the optimal architecture regarding the objective function f.

The following equations define f:

$$f(a) = \mathcal{L}_{val}(w^*(a), a) \tag{8}$$

$$w^*(a) = \arg\min_w \mathcal{L}_{train}(w, a) \tag{9}$$

For a given architecture a, the function value, $f(a)$, is an observation of the network performance with the trained weights $w^*(a)$, where the performance corresponds to the validation loss $\mathcal{L}_{val}$. The training phase, described by Equation (9), indicates that we can find $w^*(a)$ by fully training the network for a given architecture a. Training by back-propagation requires iterative gradient computation, which is computationally expensive. Thus, it is necessary to adapt performance estimation strategies for avoiding this extreme computational cost.

One possible performance estimation strategy is to consider the above as a supervised ML problem and solve it by employing a regression model. More specifically, we prepare a regression model before solving the optimization shown in Equation (7). Here, the preparation is also referred to as the initialization step[27], and it requires: 1) to collect a fixed number m of pairs $(a, f(a))$; and 2) to fit a regression model based upon this collection. The trained regression model $\hat{f}$ can provide an approximation of f. Thus, the regression model serves as a surrogate for the validation loss observation, i.e., we apply a surrogate model $\hat{f}$ which can approximate f.

When it comes to supervised learning, it is necessary to collect labeled training samples. Since such a process requires the performance evaluation of fully trained networks, the amount of labels is typically very restricted. Nevertheless, the model trained with limited data should be able to predict the performance of individual networks spreading throughout a parameter space.

To this end, we choose NGBoost[10] as the surrogate model $\hat{f}$. Different from typical regression models that return a single best guess prediction (namely, a point estimation), NGBoost allows for predictive uncertainty estimation and outputs a full probability distribution. To be more specific, NGBoost is a modular algorithm which is composed of three modular components, as shown Figure 4. The algorithm generalizes gradient boosting[28] and makes use of natural gradients and boosting to integrate three modular components: Base Learners, Parametric Probability Distribution, and Scoring Rules. In each iteration of the learning algorithm, a vector representation of the base learner's parameters (θ) for the current model input X is fed into the Distribution component, which determines then probability distribution P_θ. In the following Scoring Rules component, the scoring function S is defined based on the distribution P_θ and the prediction target y. Finally, the natural gradient of S w.r.t. θ is used to fit the Base Learners.

We select decision tree as Base Learners l, while the conditional probability in the second component follows the Normal distribution. The logarithmic scoring rule[10] is considered as S. Our choice of the NGBoost components and hyper-parameter tuning on the model follow the details used for all experiments reported in[10].

In addition, the following two crucial aspects should be considered for using the NGBoost as a surrogate model in our NAS process. Because the full training of the Transformer network is computationally very expensive, we can prepare very few samples to train the predictor described above. On the other hand, the parameter space defined in Section 2.2 is extremely large compared to the feasible number of the prepared samples. Therefore, we need to acquire the maximum information with the minimum number of samples, by spreading them out with the aim of encouraging a diversity of data. In other words, it is necessary to handle the matter of choosing sample points in the parameter space to train the surrogate, so that it has a good space-filling property. To achieve this, we apply Latin hypercube sampling (LHS)[29] when we prepare the training samples for our performance prediction model. In this sampling strategy, each sample "remembers" in which part of the search space it was taken, so that each space dimension is evenly sampled.

As another effort to improve the surrogate model, two additional parameters regarding the network are added to the vector described in Section 2.2 when we prepare the input for the surrogate model. More specifically, we concatenate the 11 integer values for the architecture parameters introduced in Table 1 with the following two values: the number of trainable parameters in the Transformer, and the Single-shot network pruning (SNIP) value[30]. The SNIP value is a pruning at initialization metric that computes a saliency metric at initialization, and it can approximate the change in loss at initialization w.r.t. removing a specific connection. This metric was originally proposed to find sparse networks, but it has been used also for estimating the performance of lightweight networks, considering the correlation between the SNIP value and the performance at initialization[27]. In order to let the model consider additional information about the performance of a given network, we feed the SNIP value to the regression model as an additional input.

2.4. Proposed algorithm

Given the backbone architecture illustrated in Section 2.1.3, we aim to find its optimal architectures that can provide better RUL prediction accuracy. This problem is formally defined as shown in Equation (7). As we mentioned earlier, we propose to reduce the computational burden required for evolutionary NAS by incorporating into the proposed algorithm the predictor introduced in Section 2.3. This idea is outlined in Algorithm 1 in the form of pseudocode, and we explain it in the following.

Our evolutionary search starts with n_{pop} randomly generated individuals, where n_{pop} denotes the population size. As shown in Figure 5, the genotype representation of each individual is an integer vector describing the architecture a, derived from the encoding specified in Section 2.2. Within the search process, the fitness of an individual is predicted by using the surrogate model $\hat{f}$, that is the probabilistic regression model defined in Section 2.3. Before entering the main loop of the search algorithm, this performance predictor must be initialized. Here, the predictor initialization step consists in training NGBoost following the procedure explained in Section 2.3. First, we sample from the search space a fixed number m of architectures, based on LHS, and prepare m pairs $(a, f(a))$ by constructing and training each sampled architecture. Note that $f(a)$ is the validation loss of a fully trained network a, and this observation of the network performance is used as the label of the training sample for the predictor. Also, note that every collected pair $(a, f(a))$ (evaluated both during the initial sampling and during the evolutionary search) is stored in memory to avoid the redundant computation for the same architecture a.

Algorithm 1 Pseudocode of the proposed algorithm.

```
1: function EVOLUTION($n_{pop}, n_{gen}$)
2:     $pop \leftarrow initialize\_pop(n_{pop})$                    ▷ $n_{pop}$: population size
3:     $\hat{f}(\cdot) \leftarrow initialize\_predictor()$          ▷ probabilistic regression model (NGBoost)
4:     $history \leftarrow Set()$                                    ▷ evaluated individuals by full training
5:     for ($gen = 0$; $gen < n_{gen}$; $gen++$) do                  ▷ $n_{gen}$: max. number of generations
6:         $evaluate\_fitness(pop, \hat{f}(ind), history)$
7:         $new\_pop \leftarrow select(pop)$
8:         $new\_pop \leftarrow crossover(new\_pop)$
9:         $new\_pop \leftarrow mutation(new\_pop)$
10:        $pop \leftarrow check\_parents(pop, new\_pop)$
11:    end for
12:    return $history$
13: end function
14:
15: procedure EVALUATE_FITNESS($pop$, $\hat{f}(\cdot)$, $history$)
16:    for $ind \in pop$ do
17:        $ind.fitness \leftarrow \hat{f}(ind)$
18:    end for
19:    $topk \leftarrow reordering(pop)$    ▷ reorder individuals w.r.t. predicted fitness and select top $k$ individuals
20:    for $ind \in topk$ do
21:        if $ind \notin history$ then
22:            $a \leftarrow phenotype\_decoding(ind)$
23:            $ind.fitness \leftarrow f(a)$                         ▷ full training to evaluate the fitness
24:            $history.add(ind)$
25:        else
26:            $ind.fitness \leftarrow history.get(ind).fitness$
27:        end if
28:    end for
29:    $retraining(\hat{f}(\cdot), topk)$                            ▷ update the predictor with the new observations
30: end procedure
```

The evolutionary process over n_{gen} generations begins with the fitness evaluation of the population. The evaluation procedure is done in two steps: 1) fitness prediction and reordering; 2) fitness observation and updating. In the first step, we predict the fitness of the individuals using the surrogate model $\hat{f}$. The computational cost of the prediction is negligible compared to the actual fitness evaluation, which requires network training. After marking each individual with the predicted fitness, we sort the individuals according to their predicted fitness, so that in the following step we can select the best individuals in terms of predicted fitness.

The ranking by the predictions can be used as it is in the evolutionary process, because the predictions given by the trained regression model broadly correlate with the actual fitness observations. However, the correlation may not be very high, since the predictor is trained on a limited number of samples evenly spread over a large search space. For this reason, we propose to observe (i.e., evaluate) only the fitness of the *elites*, i.e., the top k individuals based on the predicted fitness, that are expected to have also better actual fitness values. The additional observations are used to replace the predictions of the current elites, as well as to update the predictor. By doing so, we improve the correlation between the fitness predictions and the fitness observations, at least for the individuals that have good fitness.

As shown in Figure 5, the fitness $f(a)$ of the elites is observed by generating and training, for each of them, the Transformer network a (the phenotype) associated with its corresponding genotype. The obtained $(a, f(a))$ pairs are appended to the training data for the predictor, and the predictor is retrained. In this way, the knowledge obtained by the new observations contributes to improved fitness predictions. At the same time, those pairs are recorded in the history, which is a look-up table used to avoid redundant computation (as mentioned above). In fact, before carrying out a fitness observation on a, first we check if a exists in the history. Then, we take its fitness $f(a)$ from the history if it has already been evaluated, otherwise we perform the fitness observation for a and add the observation to the history.

After evaluating the fitness, every individual in the current population is considered to induce offspring. Specifically, the genetic operators considered in our work are crossover and mutation. Each of them is applied independently, to avoid disruptive combinations of their joint effect. First of all, reproduction starts with a custom one-point crossover that, with a probability of 0.5, mates two adjacent individuals, i.e., given that the individuals are sorted according to the fitness, the $(2i-1)$-th best individual is combined with the $(2i)$-th best individual, where $i \in [1, \frac{n_{pop}}{2}]$. The offspring population is then obtained by applying, again with a probability of 0.5, uniform mutation to the population that includes both the individuals obtained through crossover and the individuals that have not undergone crossover. During mutation, according to a probability p_{gene}, we replace the value of each gene with a uniform random value drawn between its upper and lower bounds. The value of p_{gene} is set to 0.3, so that it can lead to mutating an average of 3.3 genes (out of 11) per individual. This way, we can achieve a good compromise between excessively small or excessively disruptive mutations.

Finally, the population for the next generation is formed by the following replacement: 1) we predict the fitness of each offspring; 2) the fitness of its parents is checked; 3) if the offspring's fitness is superior compared to the fitness of one (or both) of its parents, that parent (or the one with the worst fitness, if both parents are worse than the offspring) is replaced with the offspring.

3. Results and discussion

This section presents the details of our experimentation for evaluating the proposed algorithm: first, Section 3.1 describes the CMAPSS dataset. Next, we introduce the evaluation metrics in Section 3.2. Then, the computational setup for the evolutionary runs and the details of the network training are outlined in Section 3.3, followed by the experimental results and their analysis discussed in Section 3.4.

3.1. Benchmark dataset

Accurate RUL predictions for industrial components make it possible to develop an optimal maintenance policy, which in turn contributes to reducing any unplanned downtime and cutting dispensable losses. The airline industry is a typical application scenario, since timely maintenance of aircraft engines can largely affect the overall operation cost. In fact, by predicting the engines' RUL accurately, it is possible to minimize the overall maintenance costs.

Considering the above, NASA has introduced the CMAPSS dataset, comprising several run-to-failure trajectories simulated with the CMAPSS simulator [11]. The latter simulates the degradation of a commercial turbofan engine, depicted in Figure 6, and it provides simulated sensor measurements under different settings of health-related parameters[31].

Currently, this dataset is considered one of the standard benchmarks for data-driven RUL prediction tasks. In our work, we carry out the experiments on this dataset to show fair comparisons of our method to the most recent works presented in the existing literature. The dataset contains four sub-datasets , identified as FD001-FD004. Each of them considers different operating states and fault modes, generating 21 trajectories of recordings from different sensors. The update for the RUL prediction and the sensor measurement occurs at every *cycle*, which is the time unit considered in this dataset. As summarized in Table 2, each sub-dataset comprises of a training set D_{train} and a test set D_{test}; the former provides a running history of each engine until its failure, while a history of each test engine on D_{test} ends at a certain cycle before failure. Thus, the CMAPSS dataset enables the task of accurately predicting the RUL of each test engine at the end of its given history, for which it is allowed to exploit the data in D_{train}.

The CMAPSS dataset is made available by NASA as a set of 12 plain text (ASCII) files, occupying in total 43MB. Four of these files contain the training RUL values (each file occupies less than 1KB); four other files contain the monitoring data for the training engines, measured at every cycle (with size ranging from 3.5MB for FD001 to 10.4MB for FD004); finally, the last four files contain the monitoring data for the test engines (with size ranging from 2.2MB for FD001 to 7MB for FD004).

The training RUL values are saved as a column with one integer per line, one per each training sample in the corresponding sub-dataset. The files containing the monitoring data are instead arranged in rows (one per sample) with space-separated numerical values represented as plain text with up to 4 decimal values. The full dataset is available at: `https://github.com/mohyunho/NAS_transformer/tree/main/cmapss`.

For each engine, the given multivariate time series are processed by a sliding window to prepare fixed-length sequential data, which are then used as the training and test RUL samples; the number of samples used in our experiments is described in Table 3.

We should note that the size of the dataset appears to be sufficient to train the Transformer. To verify this, we take the upper bounds of the architecture parameters (which are described in Table 1) and generate the corresponding Transformer. Then, we train the Transformer for each sub-dataset. The four training loss curves are depicted in Figure 7.

In general, FD002 and FD004 are more challenging to make precise predictions compared to FD001 and FD003, since their data are simulated under a larger number of different operating conditions.

3.2. Evaluation metrics

In our prediction tasks, the error is defined as the discrepancy between the predicted and target RUL, and d_i denotes the error on the i-th sample. The RMSE on D_{test} is then defined as follows:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} d_i^2} \tag{10}$$

where n represents the number of samples in D_{test}.

Furthermore, we consider an additional metric, called s-score, which penalizes optimistic RUL predictions. More specifically, it separates the predictions according to whether they are “optimistic” or “pessimistic” via an asymmetric function which is formulated to grant smaller values to pessimistic predictions compared to optimistic predictions. This is formulated in this way:

$$s\text{-score} = \sum_{i=1}^{n} s_i, \quad s_i = \begin{cases} e^{-\frac{d_i}{13}} - 1, & d_i < 0 \\ e^{\frac{d_i}{10}} - 1, & d_i \geq 0 \end{cases} . \tag{11}$$

Hence, different from the RMSE, the s-score evaluates the risk that the output of the network is larger than the actual RUL value.

3.3. Computational setup and training details

The Transformer network described in Section 2.1.3 is implemented in PyTorch. The baseline GA outlined in Section 2.4 is implemented using the evolutionary computation framework DEAP[32]. Our code is available online[1]. All the experiments have been carried out on a single NVIDIA Titan Xp GPU.

In the dataset, different sensors give 21 independent trajectories. Among the 21 multi-variate time series, we discard 7 sequences whose data points never vary over time and take into account the remaining 14 time series. Then, the time series data are rescaled into the range $[-1, 1]$ by using min-max normalization. For each sub-dataset, when we create samples for the Transformer networks, those normalized data are divided by applying a fixed-length time window with stride 1. The size of the windows for FD001, FD002, FD003 and FD004 is set to 40, 60, 40 and 60, respectively, following the values suggested in[22].

Our NAS algorithm involves the validation loss evaluation of the Transformer networks followed by their full training. To do so, the given D_{train} is split into D_w and D_v, where the former is a set of training purpose data and the latter is used for validation purpose: specifically, randomly chosen 80% of the engines in each sub-dataset are assigned to D_w for solving Equation (9). The remaining 20% are designated as D_v for solving Equation (8). This proportion has been determined according to the investigation conducted in our previous study[33].

The loss function we employed is the MSE, see Equation (1). The training of the network corresponds then to finding the network weights which minimize the loss by using the Adam optimizer[34]. The batch size is set to 256, and this size is also used for calculating the SNIP value when we prepare the predictor inputs. In our GA, overfitting can lead to unreliable fitness evaluations. To overcome this issue, we apply an early stopping mechanism to measure the validation loss appropriately. In detail, network training stops early in the event that the validation loss does not decrease by a given delta after a given patience; otherwise, the training continues until the predefined maximum number of epochs. The value of the delta and patience is 0.1 and 10 respectively. We train each network for at most 100 epochs on the FD001 and FD003, and 200 epochs on the FD002 and FD004. The reason for setting a larger value for the latter two sub-datasets is that they have a greater number of training samples than the former two. Thus, on FD002 and FD004, a relatively higher number of training epochs is required for allowing the learning curve to reach a plateau.

Furthermore, we apply a learning rate decay in conjunction with the early stopping mechanism; the learning rate starts with a value of 10^{-5}, and it is then multiplied by

[1]`https://github.com/mohyunho/NAS_transformer`

a factor 0.9 every 10 epochs. This contributes to suppressing the fluctuation of the validation loss curve, so that we can avoid misleading observations given by those fluctuations. One additional note is that the validation loss in our work is defined as the RMSE on D_v. As such, the fitness of each network corresponds to the validation RMSE achieved by the trained network.

3.4. Experimental results

Our experiments aim to find the optimal Transformer architectures using the evolutionary search proposed in Section 2.4, and to numerically evaluate the quality of the discovered solutions by calculating the two metrics defined in Section 3.2 on the test set D_{test} after training. Furthermore, we perform a comparative analysis contrasting the results from the proposed GA with the state-of-the-art results.

The experiments begin by initializing the predictor. The budget used to initialize the predictor, m, is determined as 100, i.e., we sample 100 different architectures by LHS, and those networks are fully trained on D_w to observe the validation RMSE on D_v. After the observations, we train the predictor to minimize the error of the regression model output. We set a small m value compared with the search space size, considering the following reasons: 1) as our predictor, we employ the NGBoost, a probabilistic regression that performs well with relatively small datasets[10]; and 2) for each generation, the predictor is updated with few samples which are expected to have better fitness.

After the initialization, we execute the proposed algorithm for a maximum number of generations n_{gen} of 10 with a population size n_{pop} of 1000. In the fitness evaluation step for each generation, we retrain at most only 1% of the population, i.e., the number of elites, k, is set to 10. Therefore, in each evolutionary run the maximum possible number of network training processes is 100 over 10 generations, but we observed that the actual number of training processes conducted in our experiments were 61, 81, 77 and 73, respectively for each sub-dataset, thanks to the history that enables to reuse the fitness values observed in the previous generations. Thus, for each evolutionary run, we merely need to train at most 200 networks (100 for the initialization and 100 for the updates) out of approximately 2.36×10^{12} possible networks in the search space. When it comes to the definition of the observation history, as discussed before this is an archive of all the solutions found during the search.

When analyzing the convergence of our GA, we consider the results from a single run of the algorithm (Section 3.4.1), whose results are then discussed in Section 3.4.2. Likewise, when comparing the quality of the obtained solutions with LHS solutions (Section 3.4.3), we define the solutions obtained by the proposed algorithm as the 10 best networks (in terms of fitness) from the history in a single run of evolutionary search. Considering multiple solutions, rather than just one best solution, also allows us to statistically compare the quality of solutions. On the contrary, when comparing our results to the state-of-the-art (Section 3.4.4), we perform 5 independent evolutionary runs, and take one best solution for each run. By considering multiple runs, we can enhance the reliability of the experimental results in terms of performance when our numerical results are compared to the results of other existing works.

3.4.1. Network specifications

The specification of each of the found solutions is outlined in Tables 4 to 7 for each sub-dataset respectively. From the specifications, we can observe that the best solutions tend to have large d_{model}, while the values of the other parameters regarding dimension do not obviously correlate with the fitness. We are able to discover that the larger number of attention heads does not always provide better fitness, as discussed in the other works[9,26]. Of note, N_{enc} and N_{dec} do not always converge to their upper bounds. This suggests that stacking identical layers largely increases computational cost, but does not always give better fitness.

3.4.2. Convergence

The convergence of each evolutionary run is analyzed by tracking the population average fitness, in conjunction with the std. dev., across 10 generations. As shown in Figure 12, for all the experiments, the average fitness decreases along with the generations. This reveals a gradual improvement in the quality of the solutions, given the fact that our algorithm can keep finding better solutions across the generations. However, while the overall trends show gradual improvements, they do not monotonically decrease. This is due to the fact that we update the predictor at every generation with new observations. A small increase in the average fitness at a certain generation indicates that the predictor provided "optimistic" fitness predictions in the previous generation, but those are then corrected by the new observations in the current generation. Additionally, we can observe that the average fitness curve becomes more flat around the last generation, which reveals that 10 generations are enough to show an improvement of the average fitness across generations. However, the std. dev. in the later generations is not clearly smaller than in the early generations. This is probably due to the fact that most fitness values in the population are predicted rather than observed.

3.4.3. Comparison with LHS

To demonstrate the advantage of our algorithm, we compare the quality of the solutions obtained with it to that of the solutions given by LHS. As discussed above, it is necessary for our algorithm to train 100 different architectures sampled by LHS to check the validation RMSE in the predictor initialization step. Since we exploit these solutions to initialize the predictor, which is only one part of our optimization algorithm, the optimal solutions found by the GA should, in principle, be better than the LHS solutions. Here, the LHS solutions are the 10 best samples out of 100 LHS samples in terms of the validation RMSE, so that we make a fair comparison with our approach by considering the same number of solutions (remember that we take the 10 best solutions from the GA). Then, we verify the advantage by comparing the solutions found by our proposed algorithm with the LHS solutions in terms of fitness as well as test performance. To be specific, we compare the quality of the solutions by means of both validation RMSE and test RMSE. These comparisons are depicted in the box plots shown in Figures 8 to 11, respectively for each of the four CMAPSS sub-datasets.

The statistical difference of the two solution sets is assessed based on the Mann-Whitney U (MWU) test[35]. For each pairwise comparison, we compute the statistical result and add a statistical annotation w.r.t. the MWU test p-value obtained on the two corresponding box plots.

Regarding the validation RMSE, which is the objective of our optimization algorithm, the proposed method has statistically better performances compared to LHS, except for FD002 where the difference is not significant ("ns"). In particular, the p-values obtained on FD001 and FD003 are lower than 0.01, showing that the results we obtained are significantly better than LHS. On FD002, even though the difference is not significant, Figure 9 graphically demonstrates that the improvement is non-trivial; instead, the difference for FD004 is significant, but as Figure 11 reveals, it has a higher p-value than that obtained for FD001 and FD003.

The proposed GA and the LHS also are also evaluated on the test set D_{test}. This performance comparison appears on the right side of Figures 8 to 11. We can visually detect that our method is better compared to LHS, in terms of test RMSE, i.e., the best solution obtained by the proposed algorithm can always provide at least one solution with lower test RMSE than any of the LHS solutions. Nevertheless, the performance differences between the 10 solutions found by the two methods are not statistically significant.

3.4.4. Comparison with the state-of-the-art

In the following, we evaluate the results obtained by our method by comparing them to the results from different methods from the most recent literature. For each sub-dataset, we execute 5 independent runs of the GA initialized with different random seeds. The multiple runs aim to enhance the reliability of the results obtained by the proposed algorithm. In detail, for each run of the evolutionary search we take the best (in terms of fitness) architecture, along with its performance. For each sub-dataset, we collect the corresponding solutions and show their performance in Tables 8 to 11. The mean and std. dev. of the 5 performance values are shown in Tables 12 and 13. The latter two tables summarize different DL based methods to be compared (including ours) and their results in terms of RMSE and s-score. For each method, the results on each sub-dataset are independent. The last column of each table contains the sum of the results across the four sub-datasets, to obtain an aggregate measure of the robustness of the compared methods.

At first, we consider the results achieved by the CNN[3], which comprises of four convolutional layers. The following RNN method[21] employs a long short-term memory (LSTM) network. Then, the next method[36] utilizes unsupervised pre-training via restricted Boltzmann machines (RBM), prior to predicting the RUL using an LSTM. The two rows below show the results from two different types of CNN-RNN models; the former, called DAG network, exploits both CNN and LSTM in parallel, with each other for extracting features[4], while the latter has a sequential architecture based on multi-head CNN followed by an LSTM[5]. The method based on an RNN autoencoder scheme[6] constructs health index curves showing degradation to predict RUL, using a bidirectional RNN based autoencoder scheme as a feature extractor. The AGCNN[7] is a custom encoder-decoder architecture in which the encoder is made up of bidirectional RNN and CNN. Lastly, the latest compared method[8] applies an attention mechanism on top of the features extracted by four convolutional layers.

Compared to the other works described above, developing our method requires less knowledge and effort from human experts, because our method automatically discovers optimal architectures for RUL prediction. In the resulting tables, each last row reports the experimental results obtained by the best solutions found with our method. With regards to the sum of test RMSE values, the proposed method outperforms the compared methods, which have all been manually developed by human experts. Furthermore, on FD002 and FD003, the RMSE results given by our proposal noticeably advance the state-of-the-art.

Furthermore, our method is evaluated with respect to the s-score defined by Equation (11). Of note, this metric is used only for test results evaluation on D_{test}; in fact, in all our experiments the RMSE is selected as the objective of our evolutionary optimization, rather than the s-score, since the RMSE is able to better guide the NAS process, compared to the s-score. In fact, following our previous study[33], when we take the RMSE rather than the s-score as a fitness function, the optimized network yields better test results in terms of both metrics.

In terms of s-score, while the proposed method does not clearly outperform state-of-the-art algorithms, its test values are comparable to the best scores in the table. Compared to other recent DL based methods, the proposed method can achieve significantly lower s-score values. Overall, although our RUL prediction model tends to provide somewhat "optimistic" predictions, it is a reliable RUL prediction tool considering its outstanding prediction accuracy.

4. Conclusions

In this paper, we introduced a NAS algorithm that employs evolutionary computation to explore the combinatorial parameter space for the Transformer architecture to be used for RUL predictions. To effectively find the optimal architecture, we developed a GA that makes use of a performance predictor. The predictor not only approximates the fitness function, but also serves to select suitable individuals to be used for retraining the predictor itself at every generation. Thanks to the surrogate model that improves over generations, our fitness predictions for the best individuals become more accurate and, ultimately, allow us to find more reliable solutions (i.e., RUL predictors).

Our algorithm was assessed on the CMAPSS dataset, that is the *de facto* standard benchmark for studies on RUL predictions. The statistical analysis revealed that the quality of the solutions found by our method is better overall than that obtained by a space-filling sampling. When we compared our results with the ones obtained by the most recent methods from the literature, we found that, in terms of performance, the proposed method is superior in terms of prediction error on the test data. Moreover, the s-score results of our method are fairly comparable with those given by the state-of-the-art methods. Overall, we demonstrated that the Transformer with optimized architecture can be a useful tool to solve the RUL prediction task. This is because it can provide highly accurate predictions, as long as its architecture is obtained by an automated architecture design process based on evolutionary computation. As a final consideration, we should point out that, while our experimentation was focused on the CMAPSS dataset due to the possibility of comparing existing methods tested on the same dataset, our proposed method is of general applicability and can be potentially applied to any other RUL prediction problem, provided that sufficient data that describe the system's dynamics and its potential faults are available.

References

[1] Zhang, W.; Yang, D.; Wang, H. Data-driven methods for predictive maintenance of industrial equipment: A survey. *IEEE Syst. J.* **2019**, *13*, 2213–2227.

[2] Atamuradov, V.; Medjaher, K.; Dersin, P.; Lamoureux, B.; Zerhouni, N. Prognostics and health management for maintenance practitioners-Review, implementation and tools evaluation. *Int. J. Progn. Health Manag.* **2017**, *8*, 1–31.

[3] Sateesh Babu, G.; Zhao, P.; Li, X.-L. Deep convolutional neural network based regression approach for estimation of remaining useful life. International Conference on Database Systems for Advanced Applications. 2016; pp 214–228.

[4] Li, J.; Li, X.; He, D. A directed acyclic graph network combined with CNN and LSTM for remaining useful life prediction. *IEEE Access* **2019**, *7*, 75464–75475.

[5] Mo, H.; Lucca, F.; Malacarne, J.; Iacca, G. Multi-Head CNN-LSTM with prediction error analysis for remaining useful life prediction. Conference of Open Innovations Association. 2020; pp 164–171.

[6] Yu, W.; Kim, I. Y.; Mechefske, C. An improved similarity-based prognostic algorithm for RUL estimation using an RNN autoencoder scheme. *Reliab. Eng. Syst. Safe.* **2020**, *199*, 106926.

[7] Liu, H.; Liu, Z.; Jia, W.; Lin, X. Remaining useful life prediction using a novel feature-attention-based end-to-end approach. *IEEE Trans. Ind. Inf.* **2020**, *17*, 1197–1207.

[8] Tan, W. M.; Teo, T. H. Remaining useful life prediction using temporal convolution with attention. *AI* **2021**, *2*, 48–70.

[9] Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; Polosukhin, I. Attention is all you need. *Adv. Neural. Inf. Process. Syst.* **2017**, *30*.

[10] Duan, T.; Anand, A.; Ding, D. Y.; Thai, K. K.; Basu, S.; Ng, A.; Schuler, A. Ngboost: Natural gradient boosting for probabilistic prediction. International Conference on Machine Learning. 2020; pp 2690–2700.

[11] Saxena, A.; Goebel, K.; Simon, D.; Eklund, N. Damage propagation modeling for aircraft engine run-to-failure simulation. International Conference on Prognostics and Health Management. 2008; pp 1–9.

[12] Dosovitskiy, A.; Beyer, L.; Kolesnikov, A.; Weissenborn, D.; Zhai, X.; Unterthiner, T.; Dehghani, M.; Minderer, M.; Heigold, G.; Gelly, S., et al. An image is worth 16x16 words: Transformers for image recognition at scale. 2020; arXiv:2010.11929.

[13] Khan, S.; Naseer, M.; Hayat, M.; Zamir, S. W.; Khan, F. S.; Shah, M. Transformers in vision: A survey. *Comput. Surv.* **2022**, *54*, 1–41.

[14] Dong, L.; Xu, S.; Xu, B. Speech-transformer: a no-recurrence sequence-to-sequence model for speech recognition. International Conference on Acoustics, Speech and Signal Processing. 2018; pp 5884–5888.

[15] Chen, N.; Watanabe, S.; Villalba, J.; Żelasko, P.; Dehak, N. Non-autoregressive transformer for speech recognition. *IEEE Signal Process Lett.* **2020**, *28*, 121–125.

[16] Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. 2018; arXiv:1810.04805.

[17] Wolf, T.; Debut, L.; Sanh, V.; Chaumond, J.; Delangue, C.; Moi, A.; Cistac, P.; Rault, T.; Louf, R.; Funtowicz, M., et al. Transformers: State-of-the-art natural language processing. Conference on Empirical Methods in Natural Language Processing: System Demonstrations. 2020; pp 38–45.

[18] Li, S.; Jin, X.; Xuan, Y.; Zhou, X.; Chen, W.; Wang, Y.-X.; Yan, X. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. *Adv. Neural. Inf. Process. Syst.* **2019**, *32*.

[19] Xu, J.; Wu, H.; Wang, J.; Long, M. Anomaly transformer: Time series anomaly detection with association discrepancy. 2021; arXiv:2110.02642.

[20] Zerveas, G.; Jayaraman, S.; Patel, D.; Bhamidipaty, A.; Eickhoff, C. A transformer-based framework for multivariate time series representation learning. Conference on Knowledge Discovery & Data Mining. 2021; pp 2114–2124.

[21] Zheng, S.; Ristovski, K.; Farahat, A.; Gupta, C. Long short-term memory network for remaining useful life estimation. International Conference on Prognostics and Health Management. 2017; pp 88–95.

[22] Zhang, Z.; Song, W.; Li, Q. Dual-Aspect Self-Attention Based on Transformer for Remaining Useful Life Prediction. *IEEE Trans. Instrum. Meas.* **2022**, *71*, 1–11.

[23] He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. Conference on Computer Vision and Pattern Recognition. 2016; pp 770–778.

[24] Ba, J. L.; Kiros, J. R.; Hinton, G. E. Layer normalization. 2016; arXiv:1607.06450.

[25] Brown, J. R.; Zhao, Y.; Shumailov, I.; Mullins, R. D. Wide Attention Is The Way Forward For Transformers. 2022; arXiv:2210.00640.

[26] Michel, P.; Levy, O.; Neubig, G. Are sixteen heads really better than one? *Adv. Neural. Inf. Process. Syst.* **2019**, *32*.

[27] White, C.; Zela, A.; Ru, R.; Liu, Y.; Hutter, F. How powerful are performance predictors in neural architecture search? *Adv. Neural. Inf. Process. Syst.* **2021**, *34*, 28454–28469.

[28] Friedman, J. H. Greedy function approximation: a gradient boosting machine. *Ann. Stat.* **2001**, 1189–1232.

[29] McKay, M. D.; Beckman, R. J.; Conover, W. J. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics* **2000**, *42*, 55–61.

[30] Lee, N.; Ajanthan, T.; Torr, P. H. Snip: Single-shot network pruning based on connection sensitivity. 2018; arXiv:1810.02340.

[31] Frederick, D. K.; DeCastro, J. A.; Litt, J. S. *User's guide for the commercial modular aero-propulsion system simulation (C-MAPSS)*; 2007.

[32] Fortin, F.-A.; De Rainville, F.-M.; Gardner, M.-A.; Parizeau, M.; Gagné, C. DEAP: Evolutionary Algorithms Made Easy. *J. Mach. Learn. Res.* **2012**, *13*, 2171–2175.

[33] Mo, H.; Custode, L. L.; Iacca, G. Evolutionary neural architecture search for remaining useful life prediction. *Appl. Soft Comput.* **2021**, *108*, 107474.

[34] Kingma, D. P.; Ba, J. Adam: A method for stochastic optimization. 2014; arXiv:1412.6980.

[35] Mann, H. B.; Whitney, D. R. On a test of whether one of two random variables is stochastically larger than the other. *Ann. Math. Stat.* **1947**, 50–60.

[36] Ellefsen, A. L.; Bjørlykhaug, E.; Æsøy, V.; Ushakov, S.; Zhang, H. Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture. *Reliab. Eng. Syst. Safe.* **2019**, *183*, 240–251.

Figures and tables

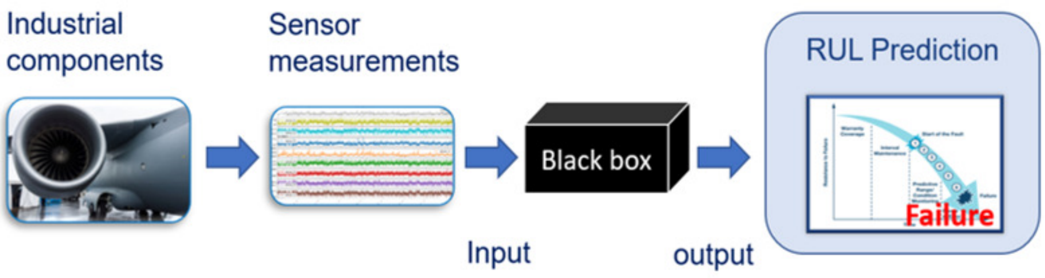

Figure 1. Flowchart of a data-driven RUL prediction task.

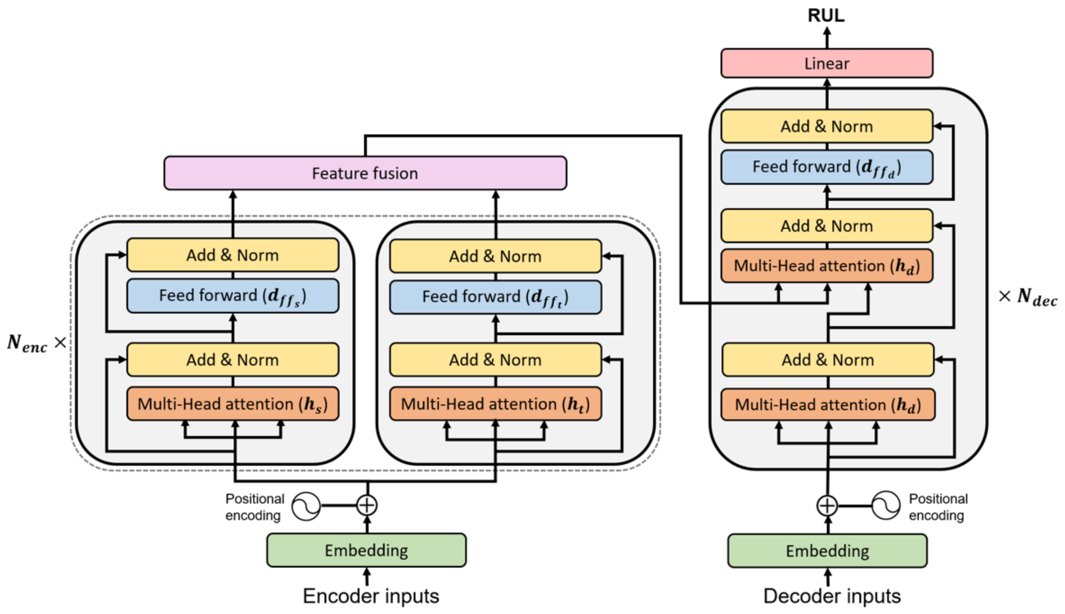

Figure 2. Model architecture of the Transformer.

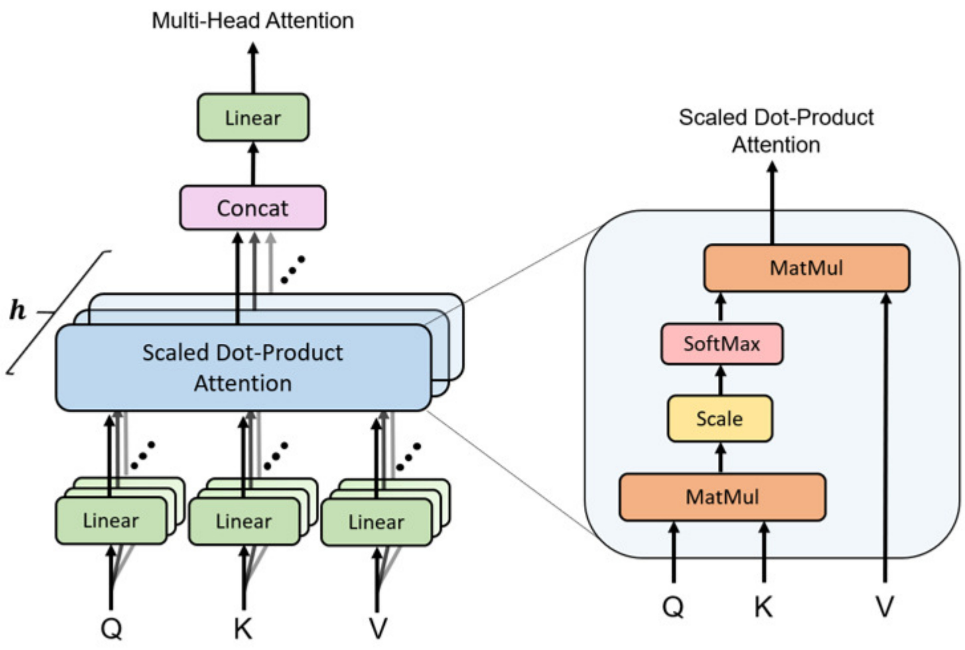

Figure 3. Multi-Head Attention based on parallel layers of Scaled Dot-Product Attention.

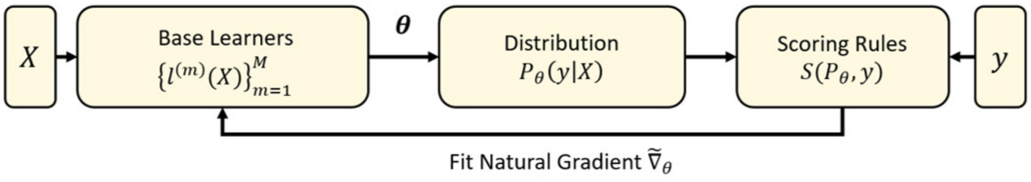
X
Base Learners $\{l^{(m)}(X)\}_{m=1}^{M}$
θ
Distribution $P_\theta(y|X)$
Scoring Rules $S(P_\theta, y)$
y
Fit Natural Gradient $\widetilde{\nabla}_\theta$

Figure 4. Overview of NGBoost which comprises of three modular components: Base Learners (l), Parametric Probability Distribution (P_θ), and Scoring Rule (S).

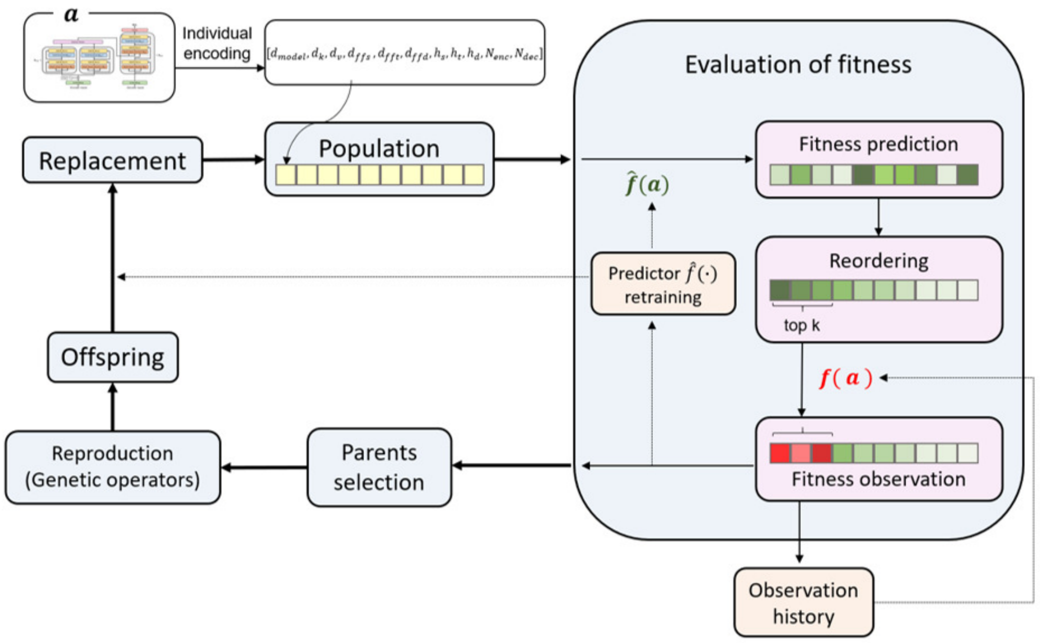

Figure 5. Overview of the proposed algorithm.

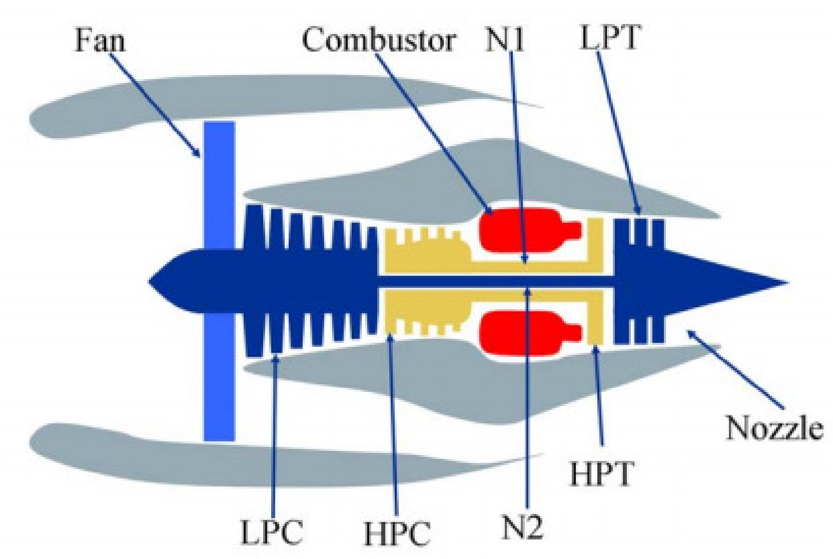

Figure 6. Simplified diagram of the turbofan engine simulated in CMAPSS[31].

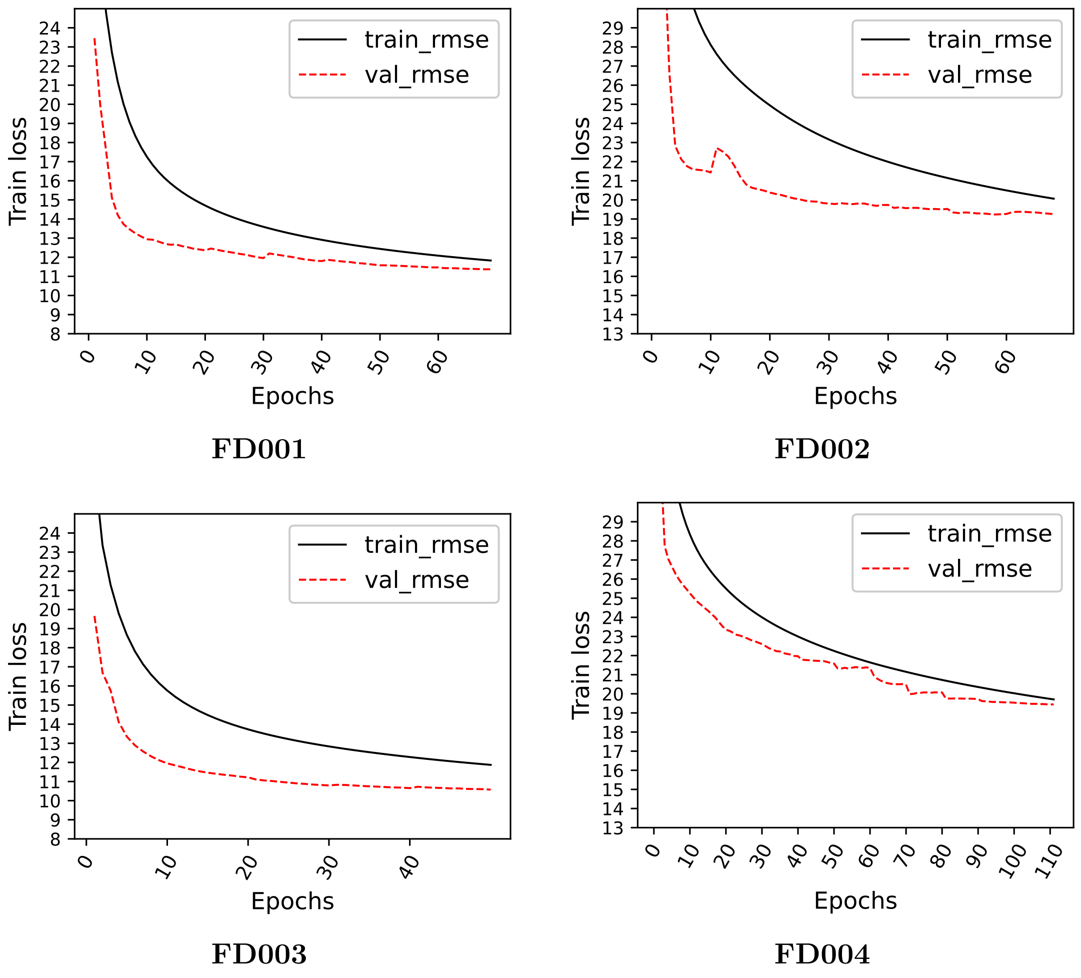

Figure 7. Training loss curves of the Transformer (based on the upper bound of each architecture parameter) on the four CMAPSS sub-datasets.

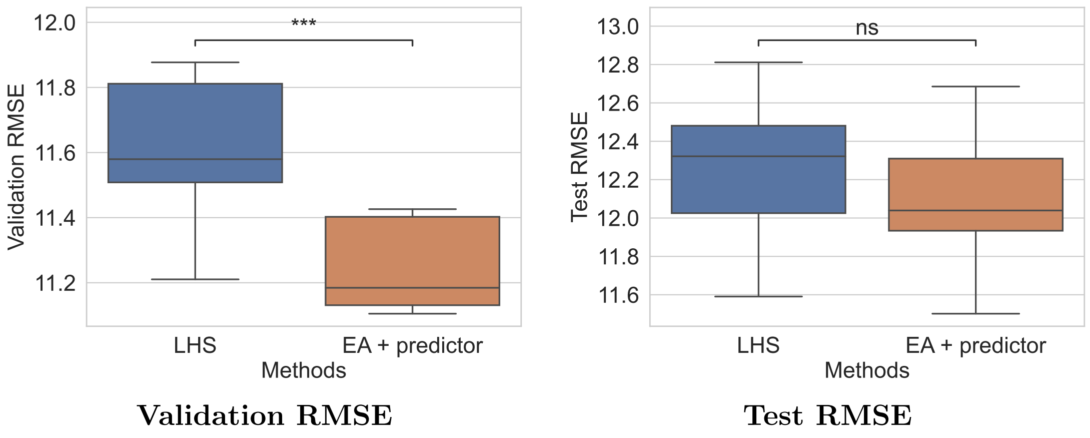

Figure 8. Box plots of the quality of solutions given by LHS and by the proposed algorithm on FD001.

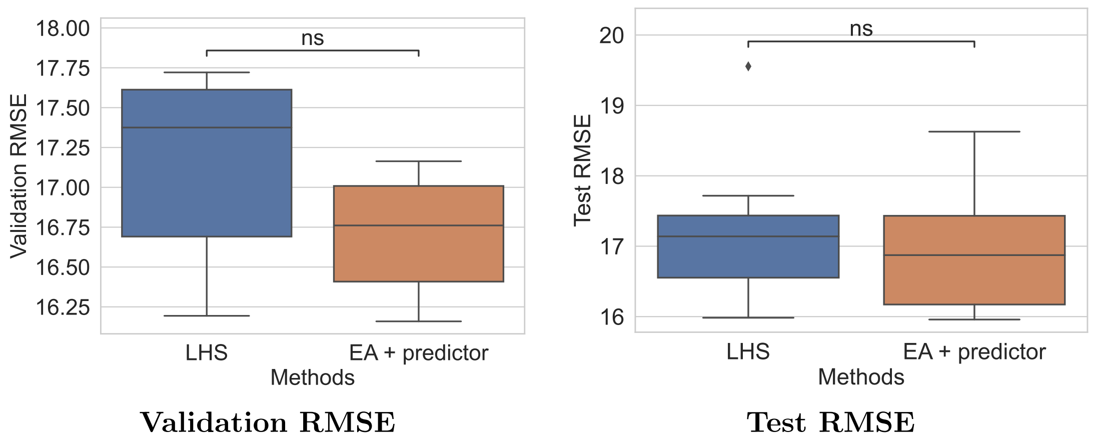

Figure 9. Box plots of the quality of solutions given by LHS and by the proposed algorithm on FD002.

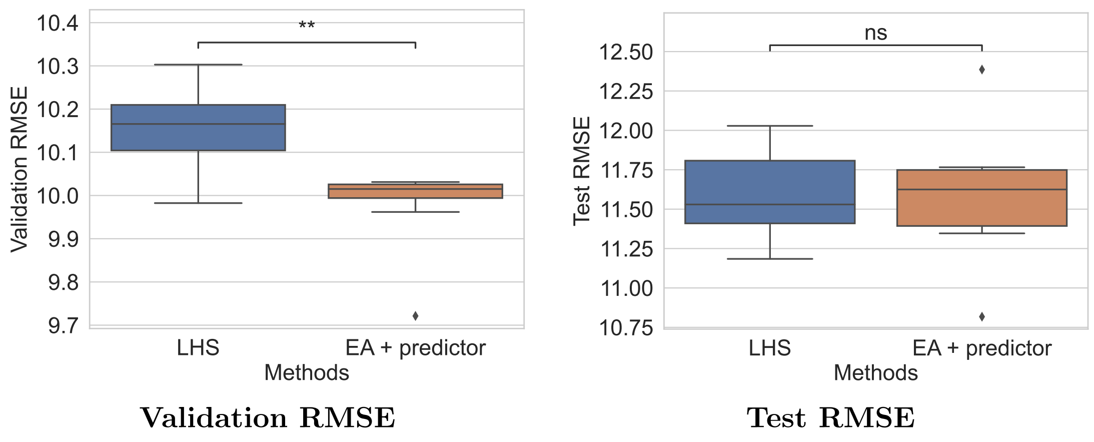

Figure 10. Box plots of the quality of solutions given by LHS and by the proposed algorithm on FD003.

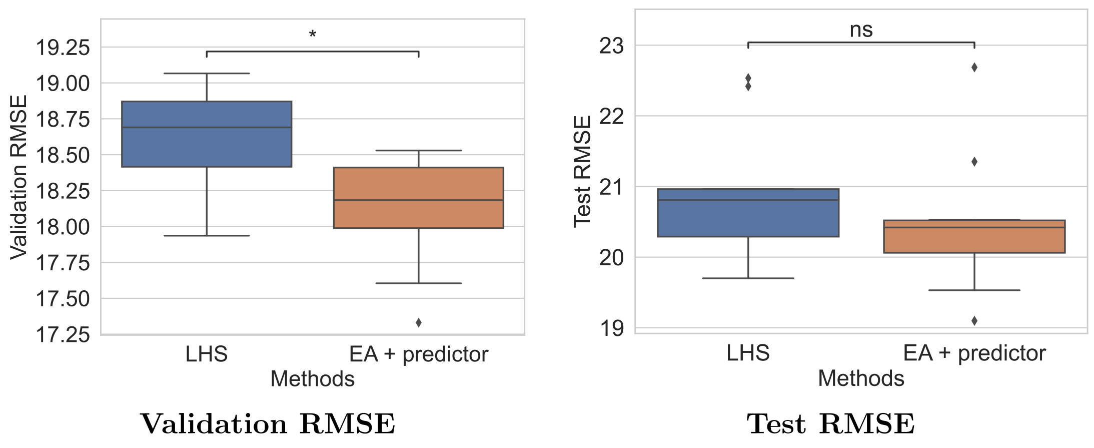

Figure 11. Box plots of the quality of solutions given by LHS and by the proposed algorithm on FD004.

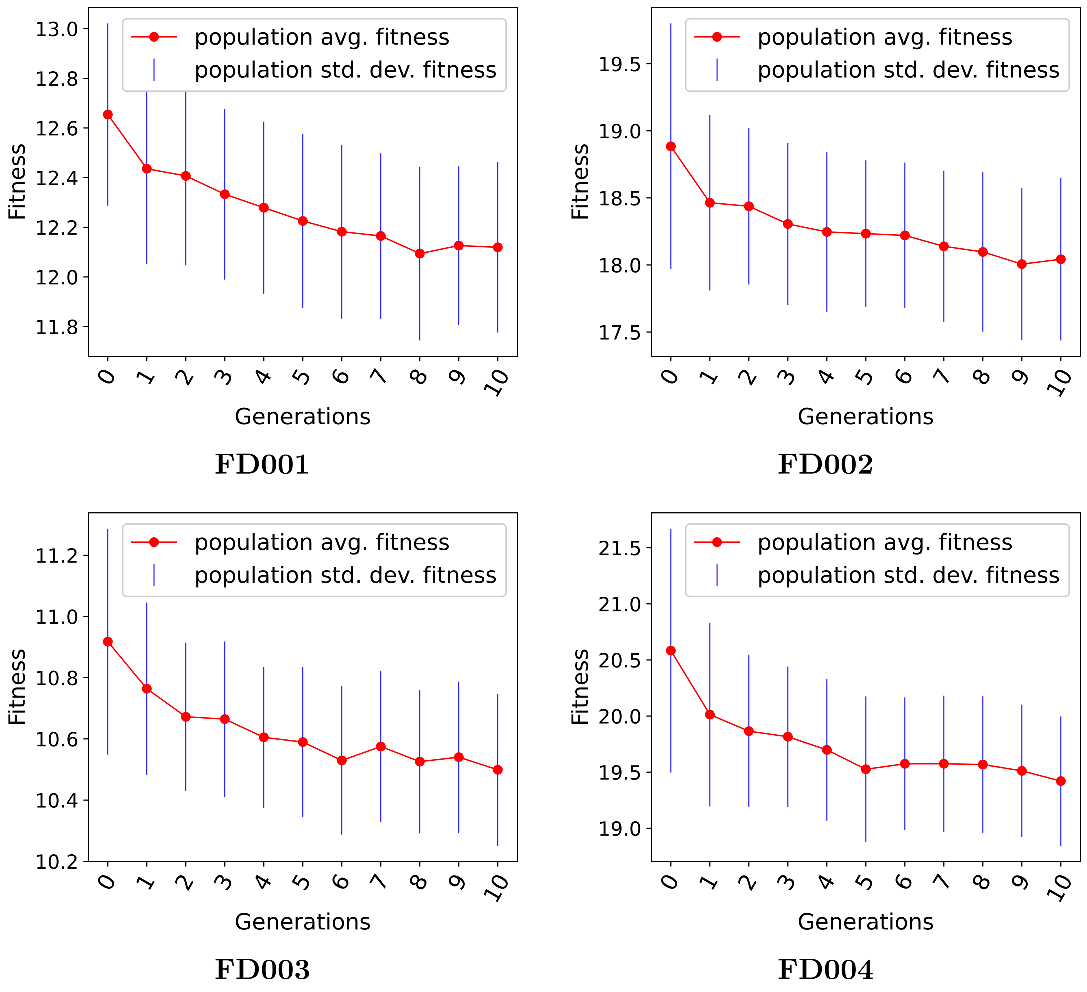

Figure 12. Average and std. dev. of the individual fitness in the population across generations for the proposed algorithm on the four CMAPSS sub-datasets.

Table 1. Architecture parameters and their bounds.

Parameter	Description	Min	Max
d_{model}	dim. of embedding and each sub-layer input/output	6	25
d_k	dim. of attention key	6	25
d_v	dim. of attention value	6	25
d_{ff_s}	dim. of FFN in sensor encoder layer	6	25
d_{ff_t}	dim. of FFN in time encoder layer	6	25
d_{ff_d}	dim. of FFN in decoder layer	6	25
h_s	number of attention heads in sensor encoder layer	1	16
h_t	number of attention heads in time encoder layer	1	16
h_d	number of attention heads in decoder layer	1	16
N_{enc}	number of encoder layers stacked	1	3
N_{dec}	number of decoder layers stacked	1	3

Table 2. CMAPSS dataset overview.

Sub-dataset	FD001	FD002	FD003	FD004
Number of engines in training set	100	260	100	249
Number of engines in test set	100	259	100	248
Max/min cycles in training set	362/128	378/128	525/145	543/128
Max/min cycles in test set	303/31	367/21	475/38	486/19
Operating conditions	1	6	1	6
Fault modes	1	1	2	2

Table 3. Number of training and test RUL samples.

Sub-dataset	FD001	FD002	FD003	FD004
Number of samples in training set (D_{train})	13702	31547	16543	36975
Number of training purpose samples (D_w)	10673	24675	12266	27392
Number of validation purpose samples (D_v)	3029	6872	4277	9583
Number of samples in test set (D_{test})	100	259	100	248

Table 4. Specification of the top 10 Transformer architectures discovered by the first run of the proposed algorithm on FD001. These are used for comparison with the LHS solutions discussed in Section 3.4.3. The boldface indicates the best architecture in terms of fitness; it corresponds to the first row of Table 8.

	Phenotype (Transformer architecture)											Performance	
Dataset	d_{model} [24, 100]	d_k [24, 100]	d_v [24, 100]	d_{ff_s} [24, 100]	d_{ff_t} [24, 100]	d_{ff_d} [24, 100]	h_s [1, 16]	h_t [1, 16]	h_d [1, 16]	N_{enc} [1, 3]	N_{dec} [1, 3]	Fitness	RMSE	s-score
FD001	80	48	92	40	24	60	16	8	7	3	1	11.43	11.51	209
	84	56	92	48	32	60	13	9	3	3	1	11.42	11.93	256
	92	52	96	60	24	96	14	12	3	2	2	11.42	12.10	257
	84	24	92	48	32	56	14	12	7	3	1	11.36	12.69	271
	100	64	92	24	28	84	14	8	15	3	1	11.21	12.58	267
	72	24	84	92	60	68	16	2	15	3	2	11.19	11.96	264
	100	48	100	40	64	60	14	6	11	3	2	11.18	12.31	282
	80	72	92	24	24	48	16	8	7	3	1	11.15	12.30	270
	100	84	72	48	28	88	13	3	8	3	3	11.14	11.98	264
	92	**24**	**84**	**84**	**24**	**80**	**16**	**4**	**14**	**3**	**3**	**11.13**	**11.50**	**202**

Table 5. Specification of the top 10 Transformer architectures discovered by the first run of the proposed algorithm on FD002. These are used for comparison with the LHS solutions discussed in Section 3.4.3. The boldface indicates the best architecture in terms of fitness; it corresponds to the first row of Table 9.

	Phenotype (Transformer architecture)											Performance	
Dataset	d_{model} [24, 100]	d_k [24, 100]	d_v [24, 100]	d_{ff_s} [24, 100]	d_{ff_t} [24, 100]	d_{ff_d} [24, 100]	h_s [1, 16]	h_t [1, 16]	h_d [1, 16]	N_{enc} [1, 3]	N_{dec} [1, 3]	Fitness	RMSE	s-score
FD002	84	88	80	32	84	40	8	11	10	3	2	17.16	17.40	1478
	100	92	44	36	52	36	14	14	3	3	2	17.09	17.80	1289
	80	64	80	52	56	48	4	12	16	2	2	17.05	17.02	1186
	100	92	44	36	100	68	11	14	16	3	3	16.87	16.73	1290
	100	84	64	96	52	36	14	14	3	3	2	16.83	18.63	1494
	96	100	36	76	40	96	1	16	9	3	2	16.70	17.44	1608
	100	40	36	64	92	68	11	14	16	1	3	16.60	16.28	1142
	96	64	36	40	92	96	14	8	16	1	2	16.35	16.11	1068
	100	88	36	100	76	80	12	10	16	1	2	16.34	15.96	1146
	96	**64**	**52**	**96**	**92**	**68**	**6**	**16**	**11**	**2**	**2**	**16.20**	**16.14**	**1131**

Table 6. Specification of the top 10 Transformer architectures discovered by the first run of the proposed algorithm on FD003. These are used for comparison with the LHS solutions discussed in Section 3.4.3. The boldface indicates the best architecture in terms of fitness; it corresponds to the first row of Table 10.

	Phenotype (Transformer architecture)											Performance	
Dataset	d_{model} [24, 100]	d_k [24, 100]	d_v [24, 100]	d_{ff_s} [24, 100]	d_{ff_t} [24, 100]	d_{ff_d} [24, 100]	h_s [1, 16]	h_t [1, 16]	h_d [1, 16]	N_{enc} [1, 3]	N_{dec} [1, 3]	Fitness	RMSE	s-score
FD003	96	92	24	84	64	24	16	16	10	3	3	10.03	11.77	252
	100	92	92	64	52	100	16	1	16	3	3	10.03	11.40	249
	84	100	64	56	100	40	10	2	7	3	2	10.03	11.69	261
	92	84	92	32	56	48	6	4	9	2	3	10.02	11.39	243
	96	96	92	64	48	60	16	10	16	1	2	10.02	12.39	301
	100	84	44	72	36	48	16	1	16	3	3	10.01	11.75	276
	92	56	92	100	84	72	9	11	13	3	3	10.00	10.82	213
	84	100	96	72	60	64	15	1	16	3	3	9.99	11.74	264
	84	92	92	100	32	56	15	5	14	3	3	9.96	11.56	243
	92	**84**	**92**	**32**	**45**	**40**	**8**	**2**	**15**	**3**	**1**	**9.72**	**11.35**	**227**

Table 7. Specification of the top 10 Transformer architectures discovered by the first run of the proposed algorithm on FD004. These are used for comparison with the LHS solutions discussed in Section 3.4.3. The boldface indicates the best architecture in terms of fitness; it corresponds to the first row of Table 11.

	Phenotype (Transformer architecture)											Performance	
Dataset	d_{model} [24, 100]	d_k [24, 100]	d_v [24, 100]	d_{ff_s} [24, 100]	d_{ff_t} [24, 100]	d_{ff_d} [24, 100]	h_s [1, 16]	h_t [1, 16]	h_d [1, 16]	N_{enc} [1, 3]	N_{dec} [1, 3]	Fitness	RMSE	s-score
FD004	76	48	80	52	96	64	4	12	16	2	2	18.53	20.50	2657
	72	80	80	92	100	64	9	9	15	3	1	18.47	21.35	3519
	92	76	76	92	64	48	5	9	15	3	1	18.44	20.23	2463
	96	88	84	48	24	60	7	8	16	3	1	18.31	20.43	2758
	92	88	68	100	32	48	5	9	15	3	1	18.31	20.41	3297
	92	100	32	72	100	36	12	16	7	3	1	18.05	19.53	3037
	84	96	52	32	88	32	1	16	16	3	1	18.00	20.53	3807
	80	88	80	52	84	32	8	14	6	3	1	17.98	22.69	7414
	92	48	32	40	100	60	12	16	7	3	1	17.60	19.10	3264
	84	**76**	**92**	**92**	**96**	**40**	**2**	**10**	**15**	**3**	**1**	**17.33**	**20.00**	**2298**

Table 8. Specification of the best architectures found in each of the 5 GA runs on FD001, in conjunction with their test RMSE and s-score performance. The mean and std. dev. of the performance reported in the table are selected for comparison to the state-of-the-art methods in Tables 12 and 13.

GA runs	Phenotype (Transformer architecture)											Fitness	Performance	
	d_{model} [24, 100]	d_k [24, 100]	d_v [24, 100]	d_{ff_s} [24, 100]	d_{ff_t} [24, 100]	d_{ff_d} [24, 100]	h_s [1, 16]	h_t [1, 16]	h_d [1, 16]	N_{enc} [1, 3]	N_{dec} [1, 3]		RMSE	s-score
1st run	92	24	84	84	24	80	16	4	14	3	3	11.13	11.50	202
2nd run	100	92	88	88	52	80	15	2	6	3	3	11.18	11.89	230
3rd run	100	92	88	24	36	72	10	15	16	3	3	10.65	11.41	193
4th run	100	48	72	40	32	60	14	10	16	3	3	11.56	11.76	236
5th run	88	28	92	40	44	68	14	5	9	3	3	10.94	11.60	217

Table 9. Specification of the best architectures found in each of the 5 GA runs on FD002, in conjunction with their test RMSE and s-score performance. The mean and std. dev. of the performance reported in the table are selected for comparison to the state-of-the-art methods in Tables 12 and 13.

GA runs	Phenotype (Transformer architecture)											Fitness	Performance	
	d_{model} [24, 100]	d_k [24, 100]	d_v [24, 100]	d_{ff_s} [24, 100]	d_{ff_t} [24, 100]	d_{ff_d} [24, 100]	h_s [1, 16]	h_t [1, 16]	h_d [1, 16]	N_{enc} [1, 3]	N_{dec} [1, 3]		RMSE	s-score
1st run	96	64	52	96	92	68	6	16	11	2	2	16.20	16.14	1131
2nd run	100	84	28	76	100	88	3	11	8	3	3	15.60	15.42	997
3rd run	92	100	28	24	80	44	8	11	12	2	3	16.87	16.26	1233
4th run	92	96	28	32	92	24	1	16	3	3	1	15.70	16.35	1163
5th run	100	92	56	68	48	72	12	4	13	2	3	16.03	15.80	1145

Table 10. Specification of the best architectures found in each of the 5 GA runs on FD003, in conjunction with their test RMSE and s-score performance. The mean and std. dev. of the performance reported in the table are selected for comparison to the state-of-the-art methods in Tables 12 and 13.

GA runs	Phenotype (Transformer architecture)											Fitness	Performance	
	d_{model} [24, 100]	d_k [24, 100]	d_v [24, 100]	d_{ff_s} [24, 100]	d_{ff_t} [24, 100]	d_{ff_d} [24, 100]	h_s [1, 16]	h_t [1, 16]	h_d [1, 16]	N_{enc} [1, 3]	N_{dec} [1, 3]		RMSE	s-score
1st run	92	84	92	32	45	40	8	2	15	3	1	9.72	11.35	227
2nd run	100	100	52	64	32	68	16	13	16	2	1	9.74	11.31	226
3rd run	88	84	68	80	44	56	14	2	15	2	2	9.77	11.15	230
4th run	96	96	92	52	48	60	16	4	11	3	3	9.64	11.39	218
5th run	88	52	92	60	88	72	12	12	12	3	3	9.63	11.57	241

Table 11. Specification of the best architectures found in each of the 5 GA runs on FD004, in conjunction with their test RMSE and s-score performance. The mean and std. dev. of the performance reported in the table are selected for comparison to the state-of-the-art methods in Tables 12 and 13.

GA runs	Phenotype (Transformer architecture)											Fitness	Performance	
	d_{model} [24, 100]	d_k [24, 100]	d_v [24, 100]	d_{ff_s} [24, 100]	d_{ff_t} [24, 100]	d_{ff_d} [24, 100]	h_s [1, 16]	h_t [1, 16]	h_d [1, 16]	N_{enc} [1, 3]	N_{dec} [1, 3]		RMSE	s-score
1st run	84	76	92	92	96	40	2	10	15	3	1	17.33	20.00	2298
2nd run	96	68	60	100	76	96	2	8	16	3	1	17.89	19.85	3038
3rd run	100	76	100	40	32	60	1	16	14	3	3	17.76	20.18	2602
4th run	100	80	24	56	84	56	3	16	5	3	2	18.27	20.70	3109
5th run	92	100	28	76	68	40	5	11	16	3	3	17.85	20.03	2315

Table 12. RUL prediction performance comparison with state-of-the-art methods (sorted by year), in terms of test RMSE. The RMSE of the proposed method is calculated as mean ± std. dev. across 5 independent runs of the evolutionary search; for each run, we take the best individual in terms of fitness and collect its performance as shown in Tables 8 to 11. The results of the compared methods are taken from the original papers, in which std. dev. is not provided. We consider the mean value in comparing our method to the others, and the number highlighted in bold indicates the best value per column.

Method	RMSE				
	FD001	FD002	FD003	FD004	Sum
CNN, 2016 [3]	18.45	30.29	19.82	29.16	97.72
LSTM, 2017 [21]	16.14	24.49	16.18	28.17	84.98
Semi-supervised DL, 2019 [36]	12.56	22.73	12.10	22.66	70.05
DAG network, 2019 [4]	11.96	20.34	12.46	22.43	67.09
Multi-head CNN-LSTM, 2020 [5]	13.27	19.49	13.21	23.89	69.86
RNN+AE, 2020 [6]	13.58	19.59	19.16	22.15	74.48
AGCNN, 2020 [7]	12.42	19.43	13.39	21.50	66.74
CNN+attention, 2021 [8]	**11.48**	17.25	12.31	20.58	61.62
Proposed method (GA+Predictor)	11.63±0.19	**15.99±0.38**	**11.35±0.15**	**20.15±0.32**	**59.12**

Table 13. RUL prediction performance comparison with state-of-the-art methods (sorted by year), in terms of s-score. The s-score of the proposed method is calculated as mean ± std. dev. across 5 independent runs of the evolutionary search; for each run, we take the best individual in terms of fitness and collect its performance as shown in Tables 8 to 11. The results of the compared methods are taken from the original papers, in which std. dev. is not provided. We consider the mean value in comparing our method to the others, and the number highlighted in bold indicates the best value per column.

Methods			s-score		
	FD001	FD002	FD003	FD004	Sum
CNN, 2016 [3]	1290	13600	1600	7890	24380
LSTM, 2017 [21]	338	4450	852	5550	11190
Semi-supervised DL, 2019 [36]	231	3370	251	2840	6692
DAG network, 2019 [4]	229	2730	553	3370	6882
Multi-head CNN-LSTM, 2020 [5]	330	2880	401	6520	10131
RNN+AE, 2020 [6]	228	2650	1727	2901	7506
AGCNN, 2020 [7]	225	1492	**227**	3392	5336
CNN+attention, 2021 [8]	**198**	1144	251	**2072**	**3665**
Proposed method (GA+Predictor)	215±18	**1133±85**	228±8	2672±386	4248