Aircraft Engine Remaining Useful Life Prediction: A Comparison Study of Kernel Adaptive Filtering Architectures

Georgios D. Karatzinis[a], Yiannis S. Boutalis[a], Steven Van Vaerenbergh[b]

[a]*Department of Electrical and Computer Engineering, Democritus University of Thrace, Xanthi, Greece*
[b]*Department of Mathematics, Statistics and Computing, University of Cantabria, Santander, Spain*

Abstract

Predicting the Remaining Useful Life (RUL) of mechanical systems poses significant challenges in Prognostics and Health Management (PHM), impacting safety and maintenance strategies. This study evaluates Kernel Adaptive Filtering (KAF) architectures for predicting the RUL of aircraft engines, using NASA's C-MAPSS dataset for an in-depth intra-comparison. We investigate the effectiveness of KAF algorithms, focusing on their performance dynamics in RUL prediction. By examining their behavior across different pre-processing scenarios and metrics, we aim to pinpoint the most reliable and efficient KAF models for aircraft engine prognostics. Further, our study extends to an inter-comparison with approximately 60 neural network approaches, revealing that KAFs outperform more than half of these models, highlighting the potential and viability of KAFs in scenarios where computational efficiency and fewer trainable parameters are both crucial. Although KAFs do not always surpass the most advanced neural networks in performance metrics, they demonstrate resilience and efficiency, particularly underscored by the ANS-QKRLS algorithm. This evaluation study offers valuable insights into KAFs for RUL prediction, highlighting their operational behavior, setting a foundation for future machine learning innovations. It also paves the way for research into hybrid models and deep-learning-inspired KAF structures, potentially enhancing prognostic tools in mechanical systems.

Keywords: Remaining Useful Life (RUL) Prediction, Kernel Adaptive Filtering (KAF), Prognostics and Health Management (PHM), C-MAPSS

1. Introduction

Kernel methods are a class of algorithms often used in machine learning [1], nonlinear signal processing and pattern analysis [2]. The core idea behind kernel methods is that, in the context of reproducing kernel Hilbert spaces (RKHS), input data are transformed into a high dimensional feature space (Hilbert space H) using a positive-definite function named *reproducing kernel* [3]. This way, the inner product operation in the feature space can be computed explicitly through a kernel evaluation. Conventional kernel-based implementations involve a wide range of batch formulations such as regularization network [4], support vector machine (SVM) [5], Gaussian process regression (GPR) [6], kernel principal component analysis (KPCA) [7], relevance vector machine (RVM) [8]. However, batch algorithms operate in an offline manner and they need to be retrained once new data are present imposing restrictions for real-time applications. Sequential learning algorithms are suitable for real-time applications as they update their parameters producing predictions, at each iteration, within a constant stream of data observations' acquisition.

Kernel adaptive filters (KAFs) [9], as a class of kernel sequential learning approaches, have received increased research interest over the past decades, mainly because of their online adaptation scheme and the universal approximation capabilities they offer. They can be considered as efficient extensions of the well-established linear adaptive filtering algorithms. Indicative applications that have been tackled using KAF variants include: stock returns prediction [10], nonlinear time series contaminated by impulsive noise [11], aero-engine degradation prediction [12], machine condition prediction [13], noisy chaotic time series prediction [14], suppressing noise and artifact interference from ECG signals [15].

Key challenges that emerge in KAFs are: i) kernel type and hyperparameters selection; ii) inherent linear growing network structure that leads to constant increase in memory and computational demands; Gaussian kernel is consid-

ered a default choice due to its universal approximation capability, while also it provides infinite dimension to the nonlinear mapping. In order to obtain a desirable approximation performance, proper specification of kernel parameters is crucial based on the under examination application. Optimal kernel parameter selection can be achieved using appropriate methods like cross validation [16], but such related methods lack in online learning due to high computational needs. More efficient online techniques for optimizing the kernel size include adaptive kernel size [17], multikernel adaptive filtering [18] and Gaussian KAFs with adaptive kernel bandwidth [19]. Online kernel-based learning using adaptive projection algorithms [20] serves as an efficient and simple structured alternative for training the model in a recursive manner. Various sparse methods have been proposed in order to address the continuously increased size and the computational cost. Widely used sparse methods in KAF context are: novelty criterion [21], surprise criterion [22], coherence criterion [23], variance criterion [24], approximate linear dependence (ALD) criterion [25] and others. Alternative approaches that curb the growth of the network are: quantization methods [26], [27], budget-based ones [28], [29], algorithms that combine established mechanisms [30], [31] and algorithmic extensions of them like nearest-instance-centroid-estimation kernel least-mean-square (NICE-KLMS) [32] and KRLS Tracker (KRLS-T) [33]. Other recent approaches that belong in the KAF family include multikernel adaptive filtering [34], Nyström kernel recursive least squares [35], kernel recursive algorithms that include M-estimate [36].

In this work, an evaluation study of KAF-based algorithms is presented which is dedicated to aircraft engine remaining useful life prediction. Remaining useful life (RUL) prediction is a real-world problem that is directly connected with the productivity and maintenance of industrial systems [37], [38] under the prognostic and health management (PHM) context. This is an imperative process to determine the time for maintenance and component replacement in industrial applications. RUL prediction spans among a plethora of problems such as rolling bearings [39], [40], Lithium-ion batteries [41], [42], supercapacitors [43], power electronics [44] and aircraft engines [45]. The under examination dataset in this paper is called C-MAPSS [46] and it has been created by NASA, simulating the degradation of aircraft engines (turbofan engines). A set of established KAF algorithms is evaluated in this RUL prediction problem in terms of divergent metrics. Also, an analysis is conducted in order to interpret the insight behavior of the mechanism that is encapsulated in each KAF algorithm. Finally, an extensive comparison is presented between KAF algorithms and neural network approaches reported in the literature in terms of performance and number of trainable weights. Summarizing, the contribution of this work is three-fold:

1. KAF Review and Algorithmic Insight - A detailed review of a subset of KAF algorithms is presented, with an emphasis on clarifying the underlying mechanisms that govern their performance. This analysis is not merely descriptive but analytical, delving into the operational details of each algorithm to understand its functional strengths and limitations within the RUL prediction problem.

2. Extensive Intra-comparison within KAF Domain - Extensive evaluation study of the adopted KAF architectures dedicated in a well-known RUL application, using different preprocessing and normalization scenarios. This is performed for *intra-comparison* purposes within KAF universe to evaluate the behavior of each KAF mechanism in the aircraft engine RUL problem and subsequently identify the most dominant in terms of performance and reliability.

3. Comprehensive Inter-comparison with Neural Networks - Comparative results with well-known neural network approaches reported in the literature (around 60 approaches) regarding aircraft engine RUL prediction problem. This is an *inter-comparison* analysis to rank KAF produced prediction behavior with other network-based approaches in terms of performance, training time and number of trainable parameters. The results demonstrate that KAF algorithms achieve decent performance with significantly fewer trainable parameters, highlighting their efficiency.

In the trade-off between computational burden and prediction accuracy, KAFs emerge as a preferable option, particularly in resource-constrained environments. Their efficiency, combined with competitive performance, paves the way for future research to explore deep learning-inspired KAF architectures. The rest of this paper is organized as follows: Section 2 details the theoretical background of Linear Adaptive Filters. Section 3 expands to Kernel Adaptive Filters, where the nonlinear extension of linear methods is explored. In Section 4, the focus shifts to sparsification with ALD, detailing the mechanisms for reducing computational complexity in kernel methods. Section 5 discusses the growing and pruning strategies, which address the challenge of managing the growth of the model size over time. Section 6 presents the quantization approaches, while candidates of combined approaches are presented in Section 7. The Remaining Useful Life problem is described in Section 8 covering the adopted data preprocessing cases and the utilized evaluation metrics. The extensive evaluation part of KAF approaches is presented in Section 9. Section 10 is devoted to the comparison of KAF algorithms with nearly 60 approaches reported in the literature, providing also a discussion analysis regarding the research outcomes of this work. Section 11 stands for the results, conclusion and

future work.

2. Linear Adaptive Filters

2.1. Least Mean Square

Least Mean Square (LMS) [47, 48] is the simplest adaptive filtering algorithm. Suppose there is a sequence of training input-output pairs $\{\boldsymbol{x}_i, y_i\}_{i=1}^{N}$, with N be the size of data, the objective is to learn a continuous input-output mapping $f : \mathbb{U} \rightarrow \mathbb{R}$. The input is assumed a L dimensional input vector and the input domain is a subspace of $\mathbb{R}^L$, i.e. $\boldsymbol{x} \in \mathbb{U} \subseteq \mathbb{R}^L$ and the output y is assumed as one dimensional, $y \in \mathbb{R}$. The LMS algorithm minimizes the cost function J_i as follows:

$$J_i = \frac{1}{2} e_i^2 \tag{1}$$

where e_i is the estimation error. The optimal weight vector at i^{th} iteration is approximately obtained by calculating:

$$\begin{aligned} e_i &= y_i - \boldsymbol{w}_{i-1}^T \boldsymbol{x}_i \\ \boldsymbol{w}_i &= \boldsymbol{w}_{i-1} + \eta e_i \boldsymbol{x}_i \end{aligned} \tag{2}$$

where η is the learning rate. The initial values of weight vector are equal to zero. The LMS algorithm is derived by using the instantaneous gradient of the cost function Eq. 1 with respect to weight vector, thus it is formulated similarly with the stochastic gradient descent algorithm (LMS is a special case of stochastic gradient descent).

2.2. Recursive Least Squares

In contrast with the LMS minimization procedure, which calculates the instantaneous value of the squared estimation error as presented in Eq. 1, the Recursive Least Squares (RLS) algorithm [49, 50] minimizes the sum of all squared estimation errors up to the current time step. Following a sequence of input-output pairs up to time $i-1$, $\{\boldsymbol{x}_j, y_j\}_{j=1}^{i-1}$, the objective is to minimize the corresponding cost:

$$\min_{w_{i-1}} \sum_{j=1}^{i-1} |y_j - \boldsymbol{x}_j^T \boldsymbol{w}_{i-1}|^2 + \lambda \|\boldsymbol{w}_{i-1}\|^2 \tag{3}$$

where λ is the regularization parameter. The recursive nature of this algorithm is reflected in the estimation of weight $\boldsymbol{w}_i$, when a new pair $\{\boldsymbol{x}_i, y_i\}$ becomes available, from the previous weight $\boldsymbol{w}_{i-1}$. Hence, the RLS algorithm iterates following the procedure:

$$\begin{aligned} r_i &= 1 + \boldsymbol{x}_i^T \boldsymbol{P}_{i-1} \boldsymbol{x}_i \\ \boldsymbol{k}_i &= \frac{\boldsymbol{P}_{i-1} \boldsymbol{x}_i}{r_i} \\ e_i &= y_i - \boldsymbol{x}_i^T \boldsymbol{w}_{i-1} \\ \boldsymbol{w}_i &= \boldsymbol{w}_{i-1} + \boldsymbol{k}_i e_i \\ \boldsymbol{P}_i &= \boldsymbol{P}_{i-1} - \frac{\boldsymbol{P}_{i-1} \boldsymbol{x}_i \boldsymbol{x}_i^T \boldsymbol{P}_{i-1}}{r_i} \end{aligned} \tag{4}$$

where $\boldsymbol{k}_i$ is the gain vector ($L \times 1$), the input $\boldsymbol{x}_i$ is also of the form ($L \times 1$), e_i is the prediction error and $\boldsymbol{P}_i$ is the correlation matrix ($L \times L$). The initial values of weight vector are equal to zero ($\boldsymbol{w}_0 = 0$), while also the initial values of correlation matrix are $\boldsymbol{P}_0 = \lambda^{-1} \boldsymbol{I}$.

3. Kernel Adaptive Filters

Kernel methods provide non-linear and non-parametric versions of conventional learning algorithms, which translate data into higher dimensional space. Then, linear adaptive algorithms are applied to transformed data, where these algorithms correspond to non-linear implementations in the original input space. The core idea behind kernel methods is that, in the context of reproducing kernel Hilbert spaces (RKHS), input data are transformed into a high dimensional feature space (Hilbert space H) using a positive-definite function named *reproducing kernel* [3]. This way, the inner product in the feature space can be computed through kernel evaluations. More specifically, there is no need to execute calculations in the high dimensional feature space as long as an approach can be expressed in terms of inner products (or kernel evaluations). In case that algorithmic operations in RKHS can be expressed by inner products, then these operations can be calculated by kernel evaluations in the input space without making any direct reference to feature vectors. This methodological aspect is commonly known as the "kernel trick" and based on this idea a wide variety of adaptive filtering algorithms has been introduced in RKHS [9].

The "kernel trick" describes that for a given algorithm that can be expressed in terms of inner products, an alternative algorithm can be constructed by replacing the inner products with a positive definite kernel function, i.e., the algorithm can be extended to RKHS. Reproducing Kernel Hilbert Spaces are Hilbert spaces that satisfy certain additional properties. RKHS theory is normally described as a transform theory between RKHSs and positive semi-definite functions called kernels. RKHS is a space of functions that: a) each point in space is a particular function; b) functions are smooth and continuous; c) linear functions in RKHS provide useful non-linear results; d) each RKHS has a unique kernel function. Each kernel induces exactly one RKHS, and each RKHS has a unique kernel function and certain problems posed in RKHSs are more easily solved by involving the kernel [51]. Based on the aforementioned properties, Kernel adaptive filters (KAF) are generated in RKHS by implementing well-known linear adaptive filtering techniques that correspond to nonlinear filters in the original input space, utilizing the linear structure and inner product of this space [17]. Therefore, the linear adaptive algorithms described in Section 2 can be extended in Kernel Least Mean Square (KLMS) [52], Kernel Recursive Least Squares (KRLS) [25] and Extended Kernel Recursive Least Squares (EX-KRLS) [53] respectively.

Based on the above rationale, suppose the goal is to learn a continuous input-output mapping $f : \mathbb{U} \rightarrow \mathbb{R}$ based on a sequence of training data $\{\boldsymbol{x}_i, y_i\}_{i=1}^{N}$, where $\mathbb{U} \subseteq \mathbb{R}^L$ is the input domain and $y \in \mathbb{R}$ is the corresponding desired output. A *kernel* (Mercer kernel) [3] is a continuous, symmetric and positive-definite function $\kappa : \mathbb{U} \times \mathbb{U} \rightarrow \mathbb{R}$, where a nonlinear mapping $\varphi(\cdot)$ is associated with this kernel to transform input data $\boldsymbol{x}$ into a potentially infinite-dimensional feature space RKHS $\mathbb{H}$. Common examples of symmetric and positive definite kernels are the linear, polynomial and Gaussian kernel as:

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \cdot \boldsymbol{x}' \tag{5}$$

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^T \cdot \boldsymbol{x}' + 1)^p \tag{6}$$

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\xi \|\boldsymbol{x} - \boldsymbol{x}'\|^2) \tag{7}$$

or equivalent for the Gaussian kernel:

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \exp\left(- \frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2h^2} \right) \tag{8}$$

where p is the order of the polynomial kernel, h is called kernel bandwidth or kernel size and ξ is the kernel parameter. From Mercer theorem [3], any reproducing kernel $\kappa(\boldsymbol{x}, \boldsymbol{x}')$ can be expressed as:

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \sum_{i=1}^{\infty} \zeta_i \phi_i(\boldsymbol{x}) \phi_i(\boldsymbol{x}') \tag{9}$$

where ζ_i and ϕ_i are the eigenvalues (non-negative) and eigenfunctions respectively. Therefore, the kernel induced mapping $\varphi(\cdot)$, where $\varphi : \mathbb{U} \rightarrow \mathbb{H}$ is constructed as:

$$\boldsymbol{\varphi}(\boldsymbol{x}) = \left[\sqrt{\zeta_1}\phi_1(\boldsymbol{x}),\ \sqrt{\zeta_2}\phi_2(\boldsymbol{x}), \dots \right] \tag{10}$$

such that:

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{\varphi}(\boldsymbol{x})^T \boldsymbol{\varphi}(\boldsymbol{x}') \tag{11}$$

Note that the dimensionality of $\mathbb{H}$ is determined by the number of strictly positive eigenvalues, which can be infinite in the case of the Gaussian kernel. Also, note that there is a unique isometric isomorphism in RKHS, which means that the linear structure and the inner product are both preserved. A linear model is constructed using the nonlinear mapping $\boldsymbol{\varphi}(\cdot)$, i.e., the kernel-induced mapping of Eq. 10 transforms the input $\boldsymbol{x}_i$ into the high dimensional feature space $\mathbb{H}$ as $\boldsymbol{\varphi}(\boldsymbol{x}_i)$, in the feature space:

$$f(\cdot) = \boldsymbol{\omega}_i^T \boldsymbol{\varphi}(\cdot) \tag{12}$$

where $\boldsymbol{\omega}_i$ is the weight vector in the feature space. Then, the learning task is to find (recursively) a weight vector at each iteration that minimizes the regularized least squares regression in $\mathbb{H}$:

$$\min_{\omega_i} \sum_{j=1}^{i} |y_j - \boldsymbol{\omega}_i^T \boldsymbol{\varphi}(\boldsymbol{x}_j)|^2 + \lambda \|\boldsymbol{\omega}_i\|^2 \tag{13}$$

Finally, by the representer theorem [54], for a new input $\boldsymbol{x}'$ the optimal solution can be expressed as:

$$\hat{f}(\boldsymbol{x}') = \sum_{j=1}^{i} \boldsymbol{\alpha}_i^j \kappa(\boldsymbol{x}_j, \boldsymbol{x}') \tag{14}$$

where $\boldsymbol{\alpha}_i^j$ is the j^{th} component of coefficient vector $\boldsymbol{\alpha}_i$ and the form of Eq. 14 reminds of a radial-basis function (RBF) network. KAF architectures create a growing RBF network which aims to [9]: i) learn the network topology, and ii) adapt the free parameters directly from data at the same time. However, two main challenges emerge in the KAF context [9]: i) growing architecture as, at each step, new input points are getting involved to the estimation leading to constant increase in memory and computational demands; ii) proper selection of Mercer kernel type and its parameters is needed. In the first case, sparsification approaches are applied to limit the expansion size of dictionary considering only the important/informative data. Such approaches are presented in Platt's novelty criterion [21] and approximate linear dependence (ALD) condition described in [25]. This way, sparsity allows the solution to be kept in memory in a compact form and to be easily assessed later rather than storing information pertaining to the entire history of training instances. Regarding the second challenge, various methods are used for kernel size selection such as cross-validation [16], adaptive width Gaussian kernel [55] and a sequential optimization strategy with adaptive kernel size that have been proposed in [17].

3.1. Kernel Least Mean Square

The implementation of Kernel Least Mean Square (KLMS) algorithm [52] starts from the fact that the output of LMS model (Eq. 2) can be expressed in terms of inner products. Thus the incorporation of kernel function is feasible, by the kernel trick, resulting in the form of Eq. 14. More specifically, the output of Eq. 2, for a given input $\boldsymbol{x}'$, can be expressed as inner product as follows:

$$\begin{aligned} f^{LMS}(\boldsymbol{x}') &= \boldsymbol{w}_i^T \boldsymbol{\varphi}(\boldsymbol{x}') \\ &= \eta \sum_{j=1}^{i} e_j \Big[\boldsymbol{\varphi}(\boldsymbol{x}_j)^T \boldsymbol{\varphi}(\boldsymbol{x}') \Big] \end{aligned} \tag{15}$$

Therefore, the output of the KLMS algorithm using the kernel trick and Eq. 11 becomes:

$$f^{KLMS}(\boldsymbol{x}') = \eta \sum_{j=1}^{i} e_j \kappa(\boldsymbol{x}_j, \boldsymbol{x}') \tag{16}$$

As can been seen from Eq. 16, the solution to the unknown nonlinear mapping is computed step-by-step leading to a growing RBF topological network. This way, the algorithm allocates a new kernel, at each time step, with center $\boldsymbol{x}'$ and fitting parameter as the measured error scaled by learning step η. Thus, a dictionary D_i is formed that increases constantly as new samples arrive. The KLMS algorithm is summarized in Algorithm 1.

Algorithm 1 Kernel Least Mean Square [52]

Input: $\{x_i, y_i\}$ *with* $i = 1 \dots N$

Initialization: choose kernel type κ , kernel parameter ξ and learning step η, $f_1 = 0$, $\boldsymbol{\alpha}_1^1 = 0$, center list $D_1 = \{\}$

for $i = 2 \dots N$ **do** ▷ Iterate over input-output pairs

  $f_{i-1}(\boldsymbol{x}_i) = \sum_{j=1}^{i-1} \boldsymbol{\alpha}_{i-1}^j \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$

  $e_i = y_i - f_{i-1}(\boldsymbol{x}_i)$ ▷ Prediction error calculation

  $D_i = \{D_{i-1}, \boldsymbol{x}_i\}$ ▷ Register new center

  $\boldsymbol{\alpha}_i^i = \eta e_i$ ▷ Store new coefficient

end for

3.2. Kernel Recursive Least Squares

Using the Mercer theorem, the input $\boldsymbol{x}_i$ can be transformed into the feature space $\mathbb{H}$ as $\boldsymbol{\varphi}(\boldsymbol{x}_i)$ and the RLS algorithm can be derived in RKHS leading to Kernel Recursive Least Squares (KRLS) algorithm [25]. Thus, for the example sequence $\{\boldsymbol{\varphi}(\boldsymbol{x}_i), y_i\}_{i=1}^N$ the Eq. 3 becomes similar with Eq. 13. Using $\boldsymbol{y}_i = [y_1, \dots, y_i]^T$ and $\boldsymbol{\Phi}_i = [\boldsymbol{\varphi}(\boldsymbol{x}_1), \dots, \boldsymbol{\varphi}(\boldsymbol{x}_i)]$ the optimal solution of weight vector is calculated generally as:

$$\boldsymbol{\omega}_i = \boldsymbol{\Phi}_i[\lambda \boldsymbol{I} + \boldsymbol{\Phi}_i^T \boldsymbol{\Phi}_i]^{-1} \boldsymbol{y}_i \tag{17}$$

The weight $\boldsymbol{\omega}$ can be also expressed explicitly as a linear combination of the input data such as $\boldsymbol{\omega}_i = \boldsymbol{\Phi}_i \boldsymbol{\alpha}_i$ with $\boldsymbol{\alpha}_i = [\lambda \boldsymbol{I} + \boldsymbol{\Phi}_i^T \boldsymbol{\Phi}_i]^{-1} \boldsymbol{y}_i$. Denoting $\tilde{\boldsymbol{K}}_i = [\lambda \boldsymbol{I} + \boldsymbol{\Phi}_i^T \boldsymbol{\Phi}_i]^{-1}$, $\tilde{\boldsymbol{K}}_i$ can be expressed in the recursive form:

$$\tilde{\boldsymbol{K}}_i = \begin{bmatrix} \tilde{\boldsymbol{K}}_{i-1} & \tilde{\boldsymbol{k}}_i \\ \tilde{\boldsymbol{k}}_i^T & \lambda + k_{ii} \end{bmatrix} \tag{18}$$

where $\tilde{\boldsymbol{k}}_i = \boldsymbol{\Phi}_{i-1}^T \boldsymbol{\varphi}(\boldsymbol{x}_i) = [\kappa(\boldsymbol{x}_i, \boldsymbol{x}_1), \dots, \kappa(\boldsymbol{x}_i, \boldsymbol{x}_{i-1})]$ is the kernel vector and $k_{ii} = \boldsymbol{\varphi}(\boldsymbol{x}_i)^T \boldsymbol{\varphi}(\boldsymbol{x}_i) = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_i)$. Thus, the inverse of $\tilde{\boldsymbol{K}}_i$ is updated using:

$$\tilde{\boldsymbol{K}}_i^{-1} = \frac{1}{\delta_i} \begin{bmatrix} \tilde{\boldsymbol{K}}_{i-1}^{-1} \delta_i + \tilde{\boldsymbol{a}}_i^T \tilde{\boldsymbol{a}}_i & -\tilde{\boldsymbol{a}}_i \\ -\tilde{\boldsymbol{a}}_i^T & 1 \end{bmatrix} \tag{19}$$

where $\tilde{\boldsymbol{a}}_i = \tilde{\boldsymbol{K}}_i^{-1} \tilde{\boldsymbol{k}}_i$ and $\delta_i = \lambda + \kappa(\boldsymbol{x}_i, \boldsymbol{x}_i) - \tilde{\boldsymbol{k}}_i^T \tilde{\boldsymbol{a}}_i$. Then, the approximated function can be expressed similarly with Eq.14 where the coefficient vector is calculated as follows:

$$\boldsymbol{\alpha}_i = \begin{bmatrix} \boldsymbol{\alpha}_{i-1} - \tilde{\boldsymbol{a}}_i \delta_i^{-1} e_i \\ \delta_i^{-1} e_i \end{bmatrix} \tag{20}$$

Algorithm 2 Kernel Recursive Least Squares [25]

Input: $\{x_i, y_i\}$ *with* $i = 1 \dots N$

Initialization: choose kernel type κ, kernel parameter ξ and regularization parameter λ, $\tilde{\boldsymbol{K}}_1^{-1} = 1/(\lambda + \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1))$, $\boldsymbol{\alpha}_1 = (\tilde{\boldsymbol{K}}_1^{-1} y_1)$, center list $D_1 = \{\}$

for $i = 2 \dots N$ **do** ▷ Iterate over input-output pairs

  $\tilde{\boldsymbol{k}}_i = [\kappa(\boldsymbol{x}_i, \boldsymbol{x}_1), \dots, \kappa(\boldsymbol{x}_i, \boldsymbol{x}_{i-1})]$ ▷ Kernel vector

  $\tilde{\boldsymbol{a}}_i = \tilde{\boldsymbol{K}}_{i-1}^{-1} \tilde{\boldsymbol{k}}_i$

  $\delta_i = \lambda + \kappa(\boldsymbol{x}_i, \boldsymbol{x}_i)) - \tilde{\boldsymbol{k}}_i^T \tilde{\boldsymbol{a}}_i$

  Compute kernel matrix $\tilde{\boldsymbol{K}}_i^{-1}$ by using Eq. 19

  $e_i = y_i - \tilde{\boldsymbol{k}}_i^T \boldsymbol{\alpha}_{i-1}$ ▷ Prediction error calculation

  Compute $\boldsymbol{\alpha}_i$ by using Eq. 20

  $D_i = \{D_{i-1}, \boldsymbol{x}_i\}$ ▷ Register new center

end for

This way, at every time-step a new unit is registered in D_i with center $\boldsymbol{x}_i$ and $\boldsymbol{\alpha}_i$ as the coefficient. More specifically, the current coefficient is calculated using $\delta_i^{-1} e_i$ (second row of Eq. 20), while the previous coefficients are updated by $-\tilde{\boldsymbol{a}}_i \delta_i^{-1} e_i$ (first row of Eq. 20). The KRLS algorithm is summarized in Algorithm 2.

4. Sparsification with ALD

As described in Section 3, at each iteration, a new feature input is registered into the current state of dictionary D_i. Adopting Gaussian kernels, a new kernel unit is registered for every new input sample as a RBF center leading to a constantly growing RBF network. This way, the network is linearly depended on the number of training samples. Sparsification techniques have been proposed in the literature to limit the expansion size of dictionary, alleviating the computational load and memory requirements. In order to curb the growth of the networks, a wide set of sparsification techniques has been proposed in the literature including novelty criterion [21], surprise criterion [22], coherence criterion [23], variance criterion [24] and others. Approximate Linear Dependence (ALD) [25] criterion is such an efficient sparsification procedure which is usually preferred. ALD selects a set of basis vectors (dictionary vectors) in the feature space resulting to a simplified network by discarding the redundant information.

More specifically, after a set of training samples $\{\boldsymbol{x}_j, y_j\}_{j=1}^{i-1}$ have been observed, ALD participates for the creation of a dictionary which is consisted of a subset of these samples, i.e. $D_{i-1} = \{\tilde{\boldsymbol{x}}_j\}_{j=1}^{m_{i-1}}$, where the feature inputs $\{\boldsymbol{\varphi}(\tilde{\boldsymbol{x}}_j)\}_{j=1}^{m_{i-1}}$ are linearly independent. Thus, every new feature input $\boldsymbol{\varphi}(\boldsymbol{x}_i)$ is tested whether it breaches (added to the dictionary as informative sample) or passes (discarded as redundant sample) the ALD condition. The ALD condition is expressed as [25]:

$$\delta_i := \min_{\boldsymbol{\alpha}_i} \left\| \sum_{j=1}^{m_{i-1}} \boldsymbol{\alpha}_j \boldsymbol{\varphi}(\tilde{\boldsymbol{x}}_j) - \boldsymbol{\varphi}(\boldsymbol{x}_i) \right\|^2 \leq \nu \tag{21}$$

where $\boldsymbol{\alpha}_i = (\alpha_1, \ldots, \alpha_{m_{i-1}})^T$ are the coefficients that satisfy the ALD condition, ν indicates the ALD threshold parameter or the level of sparsity (small positive constant), $\tilde{\boldsymbol{x}}_j$ is the j^{th} support vector in the dictionary until $i-1$ and m_{i-1} defines the number of breaches. The δ_i represents the square of the Euclidean distance between the new feature input $\boldsymbol{\varphi}(\boldsymbol{x}_i)$ and the subspace spanned by the support feature vector bases that have been already selected until time $i-1$. If Eq. 21 holds, then $\boldsymbol{\varphi}(\boldsymbol{x}_i)$ can be expressed within a squared error ν as a linear combination of current dictionary centers. In such a case, this training sample will be discarded and the expansion coefficients will not be updated. The δ_i can be expressed in terms of inner products (in $\mathbb{H}$) and subsequently as:

$$\delta_i = \min_{\boldsymbol{\alpha}} \{\boldsymbol{\alpha}^T \tilde{\boldsymbol{K}}_{i-1} \boldsymbol{\alpha} - 2\boldsymbol{\alpha}^T \tilde{\boldsymbol{k}}_{i-1}(\boldsymbol{x}_i) + k_{ii}\} \tag{22}$$

where $\tilde{\boldsymbol{k}}_{i-1}(\boldsymbol{x}_i)$ is the kernel vector with $(\tilde{\boldsymbol{k}}_{i-1}(\boldsymbol{x}_i))_l = \kappa(\tilde{\boldsymbol{x}}_l, \boldsymbol{x}_i)$, the kernel matrix is $[\tilde{\boldsymbol{K}}_{i-1}]_{l,j} = \kappa(\tilde{\boldsymbol{x}}_l, \tilde{\boldsymbol{x}}_j)$, $k_{ii} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_i)$ with $l, j = 1, \ldots, m_{i-1}$. The optimal $\tilde{\boldsymbol{a}}_i$ and the ALD threshold parameter are given by solving Eq. 22 as:

$$\tilde{\boldsymbol{a}}_i = \tilde{\boldsymbol{K}}_{i-1}^{-1} \tilde{\boldsymbol{k}}_{i-1}(\boldsymbol{x}_i), \;\; \delta_i = k_{ii} - \tilde{\boldsymbol{k}}_{i-1}(\boldsymbol{x}_i)^T \tilde{\boldsymbol{a}}_i \leq \nu \tag{23}$$

Therefore, if $\delta_i > \nu$ then ALD breaches and the current input sample, $\boldsymbol{x}_i$, is added to the dictionary so as, $D_i = D_{i-1} \cup \{\boldsymbol{x}_i\}$ and subsequently the number of breaches occurred is increased by one, $m_i = m_{i-1} + 1$. Finally, the corresponding approximation in terms of kernel matrices is $\boldsymbol{K}_i = \boldsymbol{A}_i \tilde{\boldsymbol{K}}_i \boldsymbol{A}_i^T$, where $[\boldsymbol{K}_i]_{l,j} = \kappa(\boldsymbol{x}_l, \boldsymbol{x}_j)$ with $l, j = 1, \ldots, i$ is the full kernel matrix.

4.1. The ALD-KRLS Algorithm

The ALD criterion has been initially proposed in conjunction with KRLS in [25]. The concept of this sparsification procedure defines two operational paths that are encapsulated in ALD-KRLS algorithm [25]:

- *Breach Path* ($\delta_i > \nu$): The input sample $\boldsymbol{x}_i$ is not approximately linear dependent on D_{i-1}, thus it is added to the dictionary, $D_i = D_{i-1} \cup \{\boldsymbol{x}_i\}$, and $m_i = m_{i-1} + 1$. Here, $\boldsymbol{K}_i \neq \boldsymbol{K}_{i-1}$. Then, the kernel matrix recursive formula for $\tilde{\boldsymbol{K}}_i^{-1}$ (inverse kernel matrix) and the coefficient vector $\boldsymbol{\alpha}_i$ are computed by Eq. 19 and Eq. 20 respectively as in Section 3.2 and $\boldsymbol{P}_i$ (covariance matrix) is expressed as:

$$\boldsymbol{P}_i = \begin{bmatrix} \boldsymbol{P}_{i-1} & \boldsymbol{0} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \tag{24}$$

- *Pass Path* ($\delta_i \leq \nu$): The dictionary remains unchanged, thus $D_i = D_{i-1}$, $\tilde{\boldsymbol{K}}_i^{-1} = \tilde{\boldsymbol{K}}_{i-1}^{-1}$ and $m_i = m_{i-1}$. Defining:

$$\boldsymbol{q}_i = \frac{\boldsymbol{P}_{i-1}\tilde{\boldsymbol{a}}_i}{1 + \tilde{\boldsymbol{a}}_i^T \boldsymbol{P}_{i-1}\tilde{\boldsymbol{a}}_i} \tag{25}$$

while also the recursive formula for $\boldsymbol{P}_i$:

$$\boldsymbol{P}_i = \boldsymbol{P}_{i-1} - \frac{\boldsymbol{P}_{i-1}\tilde{\boldsymbol{a}}_i\tilde{\boldsymbol{a}}_i^T \boldsymbol{P}_{i-1}}{1 + \tilde{\boldsymbol{a}}_i^T \boldsymbol{P}_{i-1}\tilde{\boldsymbol{a}}_i} \tag{26}$$

the coefficient vector $\boldsymbol{\alpha}_i$ is updated as:

$$\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i-1} + \tilde{\boldsymbol{K}}_i^{-1}\boldsymbol{q}_i(y_i - \tilde{\boldsymbol{k}}_{i-1}^T \boldsymbol{\alpha}_{i-1}) \tag{27}$$

Note that the kernel vector $\tilde{\boldsymbol{k}}_{i-1}(\boldsymbol{x}_i) = \tilde{\boldsymbol{k}}_{i-1} = \kappa(D_{i-1}, \boldsymbol{x}_i)$ is calculated using the support vectors $\tilde{\boldsymbol{x}}_j$ stored in the dictionary at each time step with the updated dictionary length m (Eq. 21). Thus, the kernel vector for the Gaussian kernel case is given by:

$$\tilde{\boldsymbol{k}}_{i-1} = \kappa(D_{i-1}, \boldsymbol{x}_i) = e^{-(\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_{1:m_{i-1}})^T(\boldsymbol{x}_i - \tilde{\boldsymbol{x}}_{1:m_{i-1}})/(2h^2)} \tag{28}$$

Also, the prediction is computed by an inner product as:

$$\hat{f}(\boldsymbol{x}_i) = \tilde{\boldsymbol{k}}_{i-1}^T \boldsymbol{\alpha}_{i-1} \tag{29}$$

Therefore, the resulting ALD-KRLS algorithm is summarized in Algorithm 3.

Algorithm 3 ALD-Kernel Recursive Least Squares [25]

Input: $\{x_i, y_i\}$ *with* $i = 1 \ldots N$

Initialization: choose kernel type κ, kernel parameter ξ and regularization parameter λ, $\tilde{\boldsymbol{K}}_1^{-1} = 1/(\lambda + \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1))$, $\boldsymbol{\alpha}_1 = (\tilde{\boldsymbol{K}}_1^{-1} y_1)$, center list $D_1 = \{\}$

for $i = 2 \ldots N$ **do** ▷ Iterate over input-output pairs
  Compute kernel vector $\tilde{\boldsymbol{k}}_i$ by using Eq. 28
  $\tilde{\boldsymbol{a}}_i = \tilde{\boldsymbol{K}}_{i-1}^{-1}\tilde{\boldsymbol{k}}_i$
  $\delta_i = \lambda + \kappa(\boldsymbol{x}_i, \boldsymbol{x}_i) - \tilde{\boldsymbol{k}}_i^T \tilde{\boldsymbol{a}}_i$
  if $\delta_i > \nu$ **then** ▷ **ALD Breach Path**
    $D_i = \{D_{i-1}, \boldsymbol{x}_i\}$ ▷ Register $\boldsymbol{x}_i$ to dictionary
    $m_i = m_{i-1} + 1$ ▷ Update number of breaches
    Compute kernel matrix $\tilde{\boldsymbol{K}}_i^{-1}$ by using Eq. 19
    Compute $\boldsymbol{P}_i$ matrix by using Eq. 24
    Compute $\boldsymbol{\alpha}_i$ by using Eq. 20
  else ▷ **ALD Pass Path** ($\delta_i \leq \nu$)
    $D_i = D_{i-1}$, $m_i = m_{i-1}$ ▷ Dictionary unchanged
    Compute $\boldsymbol{q}$ matrix by using Eq. 25
    Compute $\boldsymbol{P}_i$ matrix by using Eq. 26
    Compute $\boldsymbol{\alpha}_i$ by using Eq. 27 with $\tilde{\boldsymbol{K}}_i^{-1} = \tilde{\boldsymbol{K}}_{i-1}^{-1}$
  end if
end for

5. Sliding-Window and Fixed-Budget Kernel Algorithms

This class of kernel algorithms provides an alternative solution to sparsification procedures introducing a combination strategy of growing and pruning. This way the growing number of support vectors (directly connected with the size of kernel matrix) is curbed either by pruning the oldest input sample (sliding-window algorithm) or by pruning the least significant one (fixed-budget algorithm) at each time step.

5.1. Sliding-Window Kernel Recursive Least Squares

The Sliding Window KRLS (SW-KRLS) algorithm has been proposed in [28] combining a sliding window with fixed length and a L2 regularization. This technique presents low computational complexity and provides a solution against overfitting that tracks time variations. Also, SW-KRLS presents better performance over RLS, KRLS and ALD-KRLS when tracking abrupt changes and operating in non-stationary scenarios. However, more recent extensions achieve better tracking ability in non-stationary environments [56].

The main idea behind sliding window is that only depends on the latest m number of observations for regression. At every iteration the algorithm stores the new sample into the dictionary, while discards the most outdated (older) training sample keeping the dictionary at fixed pre-defined size. In the general form in a kernel-based least squares manner as in Eq. 17, the coefficient can be expressed as $\boldsymbol{\alpha} = [\lambda \boldsymbol{I} + \boldsymbol{\Phi}^T \boldsymbol{\Phi}]^{-1} \boldsymbol{y}$. Instead of having $\boldsymbol{K} = [\lambda \boldsymbol{I} + \boldsymbol{\Phi}^T \boldsymbol{\Phi}]^{-1}$ as in the KRLS case, here $\boldsymbol{\alpha} = (\lambda \boldsymbol{I} + \boldsymbol{K})^{-1} \boldsymbol{y}$ with $\boldsymbol{K}$ being the m by m kernel matrix. For now on, in this section $\tilde{\boldsymbol{K}}_i$ will be the regularized version of kernel matrix:

$$\tilde{\boldsymbol{K}}_i = \lambda \boldsymbol{I} + \boldsymbol{\Phi}_i^T \boldsymbol{\Phi}_i = \begin{bmatrix} \tilde{\boldsymbol{K}}_{i-1} & \tilde{\boldsymbol{k}}_{i-1}(\boldsymbol{x}_i) \\ \tilde{\boldsymbol{k}}_{i-1}(\boldsymbol{x}_i)^T & \lambda + k_{ii} \end{bmatrix} \tag{30}$$

where $k_{ii} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_i)$, λ is still the regularization factor and $\tilde{\boldsymbol{k}}_{i-1}(\boldsymbol{x}_i) = \tilde{\boldsymbol{k}}_{i-1} = [\kappa(\boldsymbol{x}_{i-m+1}, \boldsymbol{x}_i), \ldots, \kappa(\boldsymbol{x}_{i-1}, \boldsymbol{x}_i)]^T = \kappa(D_{i-1}, \boldsymbol{x}_i)$. The size of the kernel matrix remains unchanged using the aforementioned operations of growing and pruning. At time step i the new sample data is used to add one row and one column to the last row and column of the kernel matrix leading to an *upsized* matrix $\breve{\boldsymbol{K}}_i$. The inverse matrix $\breve{\boldsymbol{K}}_i^{-1}$ can be computed using the previous inverse kernel matrix $\tilde{\boldsymbol{K}}_{i-1}^{-1}$ (inverse kernel matrix before expansion-*upsizing*) by calculating:

$$\breve{\boldsymbol{K}}_i = \begin{bmatrix} \tilde{\boldsymbol{K}}_{i-1} & \boldsymbol{b} \\ \boldsymbol{b}^T & d \end{bmatrix} \Rightarrow \breve{\boldsymbol{K}}_i^{-1} = \begin{bmatrix} \tilde{\boldsymbol{K}}_{i-1}^{-1} + g\boldsymbol{e}\boldsymbol{e}^T & -g\boldsymbol{e} \\ -g\boldsymbol{e}^T & g \end{bmatrix} \tag{31}$$

where $\boldsymbol{b} = \tilde{\boldsymbol{k}}_{i-1}(\boldsymbol{x}_i)$, $d = k_{ii}$, $\boldsymbol{e} = \tilde{\boldsymbol{K}}_{i-1}^{-1} \boldsymbol{b}$ and $g = (d - \boldsymbol{b}^T \boldsymbol{e})^{-1}$. After expansion made, the kernel matrix is compressed, removing the first row and column (most outdated sample) of the *upsized* kernel matrix $\breve{\boldsymbol{K}}_i$. Thus, the *upsized* kernel matrix $\breve{\boldsymbol{K}}_i$ is *downsized* back to matrix $\tilde{\boldsymbol{K}}_i$ and its inverse matrix can be obtained based on the knowledge of $\breve{\boldsymbol{K}}_i^{-1}$ by the following calculations:

$$\breve{\boldsymbol{K}}_i = \begin{bmatrix} e & \mathbf{f}^T \\ \mathbf{f} & \boldsymbol{G} \end{bmatrix} \Rightarrow \tilde{\boldsymbol{K}}_i^{-1} = \boldsymbol{G} - \mathbf{f}\mathbf{f}^T / e \tag{32}$$

where $\tilde{\boldsymbol{K}}_i^{-1}$ can be implemented by fast matrix operations as practically the first row and column pruned, i.e., $\boldsymbol{G} \leftarrow \breve{\boldsymbol{K}}_i^{-1}(2 : |\breve{\boldsymbol{K}}_i^{-1}|, 2 : |\breve{\boldsymbol{K}}_i^{-1}|)$, $\mathbf{f} \leftarrow \breve{\boldsymbol{K}}_i^{-1}(2 : |\breve{\boldsymbol{K}}_i^{-1}|, 1)$ and $e \leftarrow \breve{\boldsymbol{K}}_i^{-1}(1, 1)$. The pruning criterion is applied when the size of the *upsized* kernel matrix $|\breve{\boldsymbol{K}}_i^{-1}|$ exceeds the memory size (dictionary size) m. Note that in this algorithm, the concept of dictionary or memory is applied also for $\boldsymbol{y}$. If the memory size exceeds m, then pruning is applied on the oldest sample pair $(\boldsymbol{x}_1, y_1)$ of memory (first row and column of the $\breve{\boldsymbol{K}}_i$ are removed). The SW-KRLS approach is summarized in Algorithm 4.

Algorithm 4 Sliding-Window Kernel Recursive Least Squares [28]

Input: $\{x_i, y_i\}$ *with* $i = 1 \ldots N$

Initialization: choose kernel type κ, kernel parameter ξ and regularization parameter λ, $\tilde{\boldsymbol{K}}_1^{-1} = 1/(\lambda + \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1))$, $\boldsymbol{\alpha}_1 = (\tilde{\boldsymbol{K}}_1^{-1} y_1)$, *memory* $= \{(\boldsymbol{x}_i, y_i)\}$

1: **for** $i = 2 \ldots N$ **do** ▷ Iterate over input-output pairs
2: Add pair $\{(\boldsymbol{x}_i, y_i)\}$ to memory and calculate $\breve{\boldsymbol{K}}_i^{-1}$ using Eq. 31 ▷ Inverse of *Upsized* kernel matrix
3: **if** memory size $> m$ **then**
4: Prune the oldest pair of data from memory and calculate $\tilde{\boldsymbol{K}}_i^{-1}$ using Eq. 32 ▷ The *upsized* kernel matrix $\breve{\boldsymbol{K}}_i$ is *downsized* back to $\tilde{\boldsymbol{K}}_i$ and its inverse matrix is obtained
5: **end if**
6: The updated solution is calculated using $\boldsymbol{\alpha} = \tilde{\boldsymbol{K}}_i^{-1} \boldsymbol{y}$
7: **end for**

5.2. Fixed-Budget Kernel Recursive Least Squares

Fixed Budget KRLS (FB-KRLS) [29] in contrast with SW-KRLS does not prune the most outdated sample. Instead, this algorithm omits the least significant center (data sample) at each time step. Also, this algorithm introduces a *label updating* procedure enhancing the tracking ability of the overall approach. More specifically, in FB-KRLS a pruning criterion is applied to determine the index j^* of the least significant stored sample among all stored j samples in memory (memory is still of size m). The utilized error is calculated by:

$$cr(\boldsymbol{x}_j, y_j) = \frac{|\boldsymbol{\alpha}_i^j|}{[\check{\boldsymbol{K}}_i^{-1}]_{j,j}} \tag{33}$$

where the index j^* is obtained as $j^* = \arg\min_{1 \leq j \leq m} \boldsymbol{cr}$. Then, the Eq. 32 is utilized as in standard SW-KRLS. Regarding the *label updating* procedure that is adopted in FB-KRLS, this phase is applied right before the *upsizing* operation (as presented in Eq. 31) to adjust the outputs stored in memory and therefore to achieve an enhanced tracking capability. The updating procedure of all the stored labels, at each step, is given by:

$$y_j \leftarrow y_j - \eta\kappa(\boldsymbol{x}_j, \boldsymbol{x}_i)(y_j - y_i), \forall j \tag{34}$$

where η is a learning parameter, j is the number of stored samples and i refers to the step when a new input-output pair sample $(\boldsymbol{x}_i, y_i)$ is received. The FB-KRLS method is summarized in Algorithm 5.

Algorithm 5 Fixed-Budget Kernel Recursive Least Squares [29]

Input: $\{x_i, y_i\}$ *with* $i = 1 \ldots N$

Initialization: choose kernel type κ, kernel parameter ξ, learning step η and regularization parameter λ, $\tilde{\boldsymbol{K}}_1^{-1} = 1/(\lambda + \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1))$, $\boldsymbol{\alpha}_1 = (\tilde{\boldsymbol{K}}_1^{-1} y_1)$, $memory = \{(\boldsymbol{x}_i, y_i)\}$

1: **for** $i = 2 \ldots N$ **do** ▷ Iterate over input-output pairs
2: Update all stored labels y_j using Eq. 34
3: Add pair $\{(\boldsymbol{x}_i, y_i)\}$ to memory and calculate $\check{\boldsymbol{K}}_i^{-1}$ using Eq. 31 ▷ Inverse of *Upsized* kernel matrix
4: **if** memory size $> m$ **then**
5: Determine least significant sample j^* using Eq. 33
6: Prune j^*th pair of data from memory and calculate $\tilde{\boldsymbol{K}}_i^{-1}$ using Eq. 32 ▷ The *upsized* kernel matrix $\check{\boldsymbol{K}}_i$ is *downsized* back to $\tilde{\boldsymbol{K}}_i$ and its inverse matrix is obtained
7: **end if**
8: The updated solution is calculated using $\boldsymbol{\alpha} = \tilde{\boldsymbol{K}}_i^{-1} \boldsymbol{y}$
9: **end for**

6. Quantization in Kernel Adaptive Filtering

As discussed in Section 4, sparsification techniques prune the growth of networks by discarding the redundant input data and accepting only the most important/informative inputs as new centers. The idea behind quantization approaches is that even if the input data are identified as redundant they can be utilized to update the coefficients of the network, then they will be discarded. These data are not important to update the structure of the network adding new centers, but they can provide a compact network form by being incorporated into the coefficients update. More specifically, the quantization approach partitions the input space into smaller regions. If the quantized input has already assigned a center, the input is identified as redundant, i.e. discarded (a new center is not registered), but the coefficient of the already registered center is updated. This way input data irrespectively of their information quality are utilized to update the coefficients of the network improving its performance. The main quantization versions in kernel adaptive filtering context are the quantized KLMS (QKLMS) [26] and quantized KRLS (QKRLS) [27] algorithms.

6.1. Quantized Kernel Least Mean Square

The QKLMS algorithm is derived directly from KLMS by applying an online vector quantization (VQ) method [26]. It is reminded that a nonlinear mapping $\boldsymbol{\varphi}(\cdot)$ transforms the input signal vector $\boldsymbol{x}_i$ into the RKHS by Mercer theorem (Eq. 11) and Eq. 12. Also, the connection between the transformed input (into the high dimensional feature space $\mathbb{H}$) $\boldsymbol{\varphi}(\boldsymbol{x}_i)$, and the kernel is defined in Eq. 11 as the usual dot product. The weight update equation in KLMS can be expressed as:

$$\boldsymbol{\omega}_i = \boldsymbol{\omega}_{i-1} + \eta e_i \boldsymbol{\varphi}(\boldsymbol{x}_i) \tag{35}$$

where $\boldsymbol{\omega}_i$ denotes the weight vector, η is the learning step and e_i is the prediction error written as:

$$e_i = y_i - \boldsymbol{\omega}_{i-1}^T \boldsymbol{\varphi}(\boldsymbol{x}_i) \tag{36}$$

By quantizing the feature vector $\boldsymbol{\varphi}(\boldsymbol{x}_i)$, the weight update equation of QKLMS algorithm is expressed as:

$$\begin{aligned} \boldsymbol{\omega}_0 &= 0 \\ e_i &= y_i - \boldsymbol{\omega}_{i-1}^T \boldsymbol{\varphi}(\boldsymbol{x}_i) \\ \boldsymbol{\omega}_i &= \boldsymbol{\omega}_{i-1} + \eta e_i \tilde{\boldsymbol{Q}}[\boldsymbol{\varphi}(\boldsymbol{x}_i)] \end{aligned} \tag{37}$$

where $\tilde{\boldsymbol{Q}}[\cdot]$ is the quantization operator in the RKHS $\mathbb{H}$. Due to the high dimentionality of the feature space, the quantization is applied in the original input space $\mathbb{U}$ instead of $\mathbb{H}$. Therefore, denoting $f_i = \boldsymbol{\omega}_i \boldsymbol{\varphi}(\cdot)$ as in Eq. 12 the QKLMS algorithm yields:

$$\begin{aligned} f_0 &= 0 \\ e_i &= y_i - f_{i-1}(\boldsymbol{x}_i) \\ f_i &= f_{i-1} + \eta e_i \kappa(\boldsymbol{Q}[\boldsymbol{x}_i], \cdot) \end{aligned} \tag{38}$$

where i is the step size, $\boldsymbol{x}_i$ is the N dimensional input, f is the input-output mapping and $\boldsymbol{Q}[\cdot]$ denotes a quantization operator in $\mathbb{U}$. The notation of both quantization operators (in feature and input space) can be simplified respectively as: $\boldsymbol{\varphi}_i^q = \tilde{\boldsymbol{Q}}[\boldsymbol{\varphi}(\boldsymbol{x}_i)]$, $\boldsymbol{x}_i^q = \boldsymbol{Q}[\boldsymbol{x}_i]$. The network size is pruned by the quantization codebook size, where the codebook C_i is equal with the dictionary D_i and the partition depends on the Euclidean distance. Note that the vector quantizer $\boldsymbol{Q}[\boldsymbol{x}_i]$ provides a mapping of the input into one of the m code (support) vectors in D, thus $D_{i-1} = \{\boldsymbol{c}_n\}_{n=1}^{m_{i-1}}$. The QKLMS algorithm is described in Algorithm 6.

Algorithm 6 Quantized Kernel Least Mean Square [26]

Input: $\{x_i, y_i\}$ *with* $i = 1 \ldots N$

Initialization: choose kernel type κ, kernel parameter ξ, learning step η, quantization threshold $\varepsilon_{\mathbb{U}} > 0$, center dictionary $D_1 = \{\boldsymbol{x}_1\}$ and coefficient vector $\boldsymbol{\alpha}_1 = [\eta y_1]$

for $i = 2 \ldots N$ **do** ▷ Iterate over input-output pairs
  $f_i = \sum_{j=1}^{size(D_{i-1})} \boldsymbol{\alpha}_{i-1}^j \kappa(D_{i-1}^j, \boldsymbol{x}_i)$ ▷ Filter output
  $e_i = y_i - f_i$ ▷ Compute error
  $dis(\boldsymbol{x}_i, D_{i-1}) = \min_{1 \le j \le size(D_{i-1})} \|\boldsymbol{x}_i - D_{i-1}^j\|$
  $j^* = \arg\min_{1 \le j \le size(D_{i-1})} \|\boldsymbol{x}_i - D_{i-1}^j\|$
  if $dis(\boldsymbol{x}_i, D_{i-1}) \le \varepsilon_{\mathbb{U}}$ **then**
    $D_i = D_{i-1}$ ▷ D_i Unchanged
    Compute $\boldsymbol{\alpha}_i^{j^*}$ by using Eq. 39 ▷ Quantize $\boldsymbol{x}_i$ to the closest center through updating the coefficient of that center
  else
    $D_i = \{D_{i-1}, \boldsymbol{x}_i\}$ ▷ Register new center
    $\boldsymbol{\alpha}_i = [\boldsymbol{\alpha}_{i-1}, \eta e_i]$ ▷ Assign new coefficient
    $m_i = m_{i-1} + 1$ ▷ Update size of Dictionary
  end if
end for

More specifically, the dictionary D_1 and the quantization size $\varepsilon_{\mathbb{U}} > 0$ are initialized. At each iteration the filter output and the error are calculated. Next, the distance between current input sample $\boldsymbol{x}_i$ and the dictionary D_{i-1} is computed producing two operational cases. In the first case if $dis(\boldsymbol{x}_i, D_{i-1}) \leq \varepsilon_{\mathbb{U}}$, the dictionary remains unchanged ($D_i = D_{i-1}$) and $\boldsymbol{x}_i$ is quantized to the closest center of the dictionary, i.e., $\boldsymbol{x}_i^q = D_{i-1}^{j^*}$ with $j^* = \arg\min_{1 \leq j \leq size(D_{i-1})} \|\boldsymbol{x}_i - D_{i-1}^j\|$. The coefficient of the closest center is updated as:

$$\boldsymbol{\alpha}_i^{j^*} = \boldsymbol{\alpha}_{i-1}^{j^*} + \eta e_i \tag{39}$$

Otherwise, in the second case where $dis(\boldsymbol{x}_i, D_{i-1}) > \varepsilon_{\mathbb{U}}$, the dictionary is updated and new coefficient is assigned. Finally, the output of the QKLMS can be expressed similar with Eq. 14 as:

$$\hat{f}(\boldsymbol{x}_i) = \sum_{j=1}^{m} \boldsymbol{\alpha}_{i-1}^j \kappa(D_{i-1}^j, \boldsymbol{x}_i) = \boldsymbol{\alpha}_{i-1} \tilde{\boldsymbol{k}}_i^T \tag{40}$$

where m is the size of the dictionary i.e., $m = size(D_{i-1})$, the coefficient vector in RKHS is $\boldsymbol{\alpha}_{i-1} = [\boldsymbol{\alpha}_{i-1}^1, \boldsymbol{\alpha}_{i-1}^2, \ldots, \boldsymbol{\alpha}_{i-1}^m]$ and $\tilde{\boldsymbol{k}}_i = [\kappa(D_{i-1}^1, \boldsymbol{x}_i), \kappa(D_{i-1}^2, \boldsymbol{x}_i), \ldots, \kappa(D_{i-1}^m, \boldsymbol{x}_i)]$ or simply $\tilde{\boldsymbol{k}}_i = \kappa(D_{i-1}, \boldsymbol{x}_i)$ as D_{i-1} contains the support vectors (subset of input samples) $\tilde{\boldsymbol{x}}_j$ until $i-1$ and is composed of m quantization regions and D_{i-1}^j is the j^{th} member of dictionary D_{i-1}.

6.2. Quantized Kernel Recursive Least Squares

The Quantized Kernel Recursive Least Squares (QKRLS) algorithm has been proposed in [27] deriving a quantization approach of KRLS replacing the original input with the quantized one. As in the QKLMS case, the dictionary D_i at time step i is composed of m code (support) vectors, i.e., $D_i = \{\boldsymbol{c}_n \in \mathbb{U}\}_{n=1}^{m_{i-1}}$. The cost function can be expressed as:

$$\sum_{n=1}^{m} \Big(\sum_{j=1}^{L_n} [y_{nj} - \boldsymbol{\omega}^T \varphi(\boldsymbol{c}_n)]^2 \Big) + \lambda \|\boldsymbol{\omega}\|^2 \tag{41}$$

where λ is still the regularization parameter, L_n denotes the number of input data that lie in the same quantization region with its center $\boldsymbol{c}_n$ and y_{nj} is the j^{th} desired output within the n^{th} quantization region. The optimal solution, as a quantized version of Eq. 17, is given by:

$$\boldsymbol{\omega}_i = [\tilde{\boldsymbol{\Phi}}_i \boldsymbol{\Lambda}_i \tilde{\boldsymbol{\Phi}}_i^T + \lambda \boldsymbol{I}]^{-1} \tilde{\boldsymbol{\Phi}}_i \tilde{\boldsymbol{y}}_i \tag{42}$$

where $\tilde{\boldsymbol{\Phi}}_i = [\varphi(\boldsymbol{c}_1), \ldots, \varphi(\boldsymbol{c}_m)]$ denoting the centers in the RKHS, $\boldsymbol{\Lambda}_i = \text{diag}[L_1, \ldots, L_m]$ is a diagonal matrix, $\tilde{\boldsymbol{y}}_i$ is the vector of quantized desired outputs $\tilde{\boldsymbol{y}}_i = [\sum_{j=1}^{L_1} y_{1j}, \ldots, \sum_{j=1}^{L_m} y_{mj}]^T$. Then, denoting $\boldsymbol{P}_i = [\boldsymbol{\Lambda}_i \tilde{\boldsymbol{K}}_i + \lambda \boldsymbol{I}]^{-1}$ with $\tilde{\boldsymbol{K}}_i = \tilde{\boldsymbol{\Phi}}_i^T \tilde{\boldsymbol{\Phi}}_i$ the optimal solution becomes $\boldsymbol{\omega}_i = \tilde{\boldsymbol{\Phi}}_i \boldsymbol{\alpha}_i$. Similar with the QKLMS algorithm, two operational cases emerge regarding the distance between x_i and the dictionary D_{i-1}:

- If distance meets the condition $dis(\boldsymbol{x}_i, D_{i-1}) \leq \varepsilon_{\mathbb{U}}$ the dictionary will remain unchanged ($D_i = D_{i-1}$ and $\tilde{\boldsymbol{K}}_i = \tilde{\boldsymbol{K}}_{i-1}$) and the input $\boldsymbol{x}_i$ is quantized to the j^*th element (closest center) of the dictionary, i.e., $D_{i-1}^{j^*} = \boldsymbol{Q}[\boldsymbol{x}_i]$ where $j^* = \arg\min_{1 \leq j \leq size(D_{i-1})} \|\boldsymbol{x}_i - D_{i-1}^j\|$. Therefore, in this case distance is $dis(\boldsymbol{x}_i, D_{i-1}) = \|\boldsymbol{x}_i - D_{i-1}^{j^*}\|$. The updated forms of $\boldsymbol{\Lambda}_i$ and $\tilde{\boldsymbol{y}}_i$ are given by:

$$\begin{aligned} \boldsymbol{\Lambda}_i &= \boldsymbol{\Lambda}_{i-1} + \theta_{j^*} \theta_{j^*}^T \\ \tilde{\boldsymbol{y}}_i &= \tilde{\boldsymbol{y}}_{i-1} - y_i \theta_{j^*} \end{aligned} \tag{43}$$

  where θ_{j^*} is a column vector of size equal with dictionary ($|D_{i-1}|$) with the j^*th element being 1 and all other entries being 0. The matrix $\boldsymbol{P}_i$ can be expressed as $\boldsymbol{P}_i = [\boldsymbol{P}_{i-1}^{-1} + \theta_{j^*} \theta_{j^*}^T \tilde{\boldsymbol{K}}_{i-1}]^{-1}$ which can be rewritten, by the matrix inversion lemma, as:

$$\boldsymbol{P}_i = \boldsymbol{P}_{i-1} - \frac{\boldsymbol{P}_{i-1}^{j^*} [(\tilde{\boldsymbol{K}}_{i-1}^{j^*})^T \boldsymbol{P}_{i-1}]}{1 + (\tilde{\boldsymbol{K}}_{i-1}^{j^*})^T \boldsymbol{P}_{i-1}^{j^*}} \tag{44}$$

where $\boldsymbol{P}_{i-1}^{j^*}$ and $\tilde{\boldsymbol{K}}_{i-1}^{j^*}$ denote the j^*th columns of $\boldsymbol{P}_{i-1}$ and $\tilde{\boldsymbol{K}}_{i-1}$ respectively. Then, the coefficient vector can be expressed as:

$$\boldsymbol{\alpha}_i = \boldsymbol{P}_i \tilde{\boldsymbol{y}}_i = \boldsymbol{\alpha}_{i-1} + \frac{\boldsymbol{P}_{i-1}^{j^*}[y_i - (\tilde{\boldsymbol{K}}_{i-1}^{j^*} =^T \boldsymbol{\alpha}_{i-1}]}{1 + (\tilde{\boldsymbol{K}}_{i-1}^{j^*})^T \boldsymbol{P}_{i-1}^{j^*}} \tag{45}$$

- If distance meets the condition $dis(\boldsymbol{x}_i, D_{i-1}) > \varepsilon_{\mathbb{U}}$ the dictionary is updated $D_i = \{D_{i-1}, \boldsymbol{x}_i\}$. In this case, $\boldsymbol{\Lambda}_i$ and $\tilde{\boldsymbol{y}}_i$ are given by:

$$\boldsymbol{\Lambda}_i = \begin{bmatrix} \boldsymbol{\Lambda}_{i-1} & \\ & 1 \end{bmatrix}, \tilde{\boldsymbol{y}}_i \quad = \begin{bmatrix} \tilde{\boldsymbol{y}}_{i-1} \\ y_i \end{bmatrix} \tag{46}$$

The matrix $\boldsymbol{P}_i$ can be expressed as:

$$\boldsymbol{P}_i = r_i^{-1} \begin{bmatrix} \boldsymbol{P}_{i-1} r_i + z_i^{\Lambda} {z_i}^T & -z_i^{\Lambda} \\ -{z_i}^T & 1 \end{bmatrix} \tag{47}$$

where $r_i = \lambda + \kappa(\boldsymbol{x}_i, \boldsymbol{x}_i) - {\boldsymbol{h}_i}^T z_i^{\Lambda}$, $z_i^{\Lambda} = \boldsymbol{P}_{i-1}\boldsymbol{\Lambda}_{i-1}\boldsymbol{h}_i$, $\boldsymbol{h}_i = [\kappa(D_{i-1}^1, \boldsymbol{x}_i), \ldots, \kappa(D_{i-1}^m, \boldsymbol{x}_i)]^T$ with m being the size of dictionary at $i-1$ and $z_i = \boldsymbol{P}_{i-1}\boldsymbol{h}_i$. Thus, the coefficient vector is obtained by:

$$\boldsymbol{\alpha}_i = \boldsymbol{P}_i \tilde{\boldsymbol{y}}_i = \begin{bmatrix} \boldsymbol{\alpha}_{i-1} - z_i^{\Lambda} r_i^{-1} e_i \\ r_i^{-1} e_i \end{bmatrix} \tag{48}$$

where the prediction error is $e_i = y_i - {\boldsymbol{h}_i}^T \boldsymbol{\alpha}_{i-1}$.

The QKRLS algorithm is summarized in Algorithm 7.

Algorithm 7 Quantized Kernel Recursive Least Squares [27]

Input: $\{x_i, y_i\}$ *with* $i = 1 \ldots N$

Initialization: choose kernel type κ, kernel parameter ξ, learning step η, quantization threshold $\varepsilon_{\mathbb{U}} > 0$, center dictionary $D_1 = \{\boldsymbol{x}_1\}$, $\Lambda_1 = 1$, $\boldsymbol{P}_1 = [\lambda + \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1)]^{-1}$ and coefficient vector $\boldsymbol{\alpha}_1 = \boldsymbol{P}_1 y_1$

for $i = 2 \ldots N$ **do** ▷ Iterate over input-output pairs
  Compute distance between $\boldsymbol{x}_i$ and D_{i-1}: $dis(\boldsymbol{x}_i, D_{i-1}) = \min_{1 \le j \le size(D_{i-1})} \|\boldsymbol{x}_i - D_{i-1}^j\|$
  $j^* = \arg\min_{1 \le j \le size(D_{i-1})} \|\boldsymbol{x}_i - D_{i-1}^j\|$
  if $dis(\boldsymbol{x}_i, D_{i-1}) \le \varepsilon_{\mathbb{U}}$ **then**
    $D_i = D_{i-1}, m_i = m_{i-1}$ ▷ D_i Unchanged
    Update $\boldsymbol{\Lambda}_i$ using Eq. 43
    Update $\boldsymbol{P}_i$ using Eq. 44
    Compute $\boldsymbol{\alpha}_i^{j^*}$ by using Eq. 45 ▷ Quantize $\boldsymbol{x}_i$ to the closest center through updating the coefficient of that center
  else
    $D_i = \{D_{i-1}, \boldsymbol{x}_i\}$ ▷ Register new center
    Update $\boldsymbol{\Lambda}_i$ using Eq. 46
    Update $\boldsymbol{P}_i$ using Eq. 47
    Update $\boldsymbol{\alpha}_i$ using Eq. 48 ▷ Quantize $\boldsymbol{x}_i$ to itself
    $m_i = m_{i-1} + 1$ ▷ Update size of Dictionary
  end if
end for

7. Combined Approaches

Combination of already established mechanisms in the KAF domain has been motivated by the increasing computational complexity and memory requirements. The increase of data size expands the computational complexity regarding the calculation of kernel inverse matrix. Thus, this challenge leads to the development of combined algorithmic extensions embodying sparsification approaches with the quantization method. In this section, two of these implementation will be described.

7.1. QALD-KRLS

This algorithm, QALD-KRLS [30], combines the ALD sparsification criterion with the quantization technique reducing the kernel structure size. This combination is implemented by the resetting occurred in line 12 of Algorithm 8. In [30], the authors highlight that this resseting has no effect on the operation of the previous mapping procedure. Parameter θ_{j^*} is a column vector of size equal with dictionary ($|D_{i-1}|$) with the j^*th element being 1 and all other entries being 0 (as presented in the QKRLS case). In rest of the algorithmic procedure the principal matrices and vectors are calculated as in the ALD-KRLS case.

Algorithm 8 QALD-KRLS [30]

Input: $\{x_i, y_i\}$ *with* $i = 1 \dots N$

Initialization: choose kernel type κ, kernel parameter ξ and regularization parameter λ, $\tilde{\boldsymbol{K}}_1^{-1} = 1/(\lambda + \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1))$, $\boldsymbol{\alpha}_1 = (\tilde{\boldsymbol{K}}_1^{-1} y_1)$, center list $D_1 = \{\}$

1: **for** $i = 2 \dots N$ **do** ▷ Iterate over input-output pairs
2: Compute kernel vector $\tilde{\boldsymbol{k}}_i$ by using Eq. 28
3: $\tilde{\boldsymbol{a}}_i = \tilde{\boldsymbol{K}}_{i-1}^{-1} \tilde{\boldsymbol{k}}_i$
4: $\delta_i = \lambda + \kappa(\boldsymbol{x}_i, \boldsymbol{x}_i) - \tilde{\boldsymbol{k}}_i^T \tilde{\boldsymbol{a}}_i$
5: Compute distance between $\boldsymbol{x}_i$ and D_{i-1}: $dis(\boldsymbol{x}_i, D_{i-1}) = \min_{1 \le j \le size(D_{i-1})} \|\boldsymbol{x}_i - D_{i-1}^j\|$
6: **if** $\delta_i > \nu$, $dis(\boldsymbol{x}_i, D_{i-1}) > \varepsilon_{\mathbb{U}}$ **then**
7: $D_i = \{D_{i-1}, \boldsymbol{x}_i\}$ ▷ Register $\boldsymbol{x}_i$ to dictionary
8: $m_i = m_{i-1} + 1$ ▷ Update number of breaches
9: Compute kernel matrix $\tilde{\boldsymbol{K}}_i^{-1}$, $\boldsymbol{P}_i$ and $\boldsymbol{\alpha}_i$ as in the ALD-KRLS case by using Eq. 19, 24, 20
10: **else**
11: **if** $dis(\boldsymbol{x}_i, D_{i-1}) \le \varepsilon_{\mathbb{U}}$ **then**
12: **reset:** $\tilde{\boldsymbol{a}}_i = \theta_{j^*}$
13: **end if**
14: $D_i = D_{i-1}$, $m_i = m_{i-1}$ ▷ Dictionary unchanged
15: Compute $\boldsymbol{q}$, $\boldsymbol{P}_i$ and $\boldsymbol{\alpha}_i$ as in the ALD-KRLS by using Eq. 25, Eq. 26 and Eq. 27 with $\tilde{\boldsymbol{K}}_i^{-1} = \tilde{\boldsymbol{K}}_{i-1}^{-1}$
16: **end if**
17: **end for**

7.2. ANS-QKRLS

The adaptive normalized sparse QKRLS algorithm (ANS-QKRLS) [31] is a combination of four main components in order to bring computational efficiency up and enhance the tracking capability of KRLS in time-varying environments. These combined components are: a) ALD sparsification criterion; b) coherence sparsification criterion; c) quantization, and; d) dynamic adjustment of weights; More specifically, the first two components are combined to detect the contribution of input to the current dictionary at each time step. This way, the informative data are registered when the contribution is higher than specific thresholds reducing the redundant information, while at the same time the dimension of kernel matrix is effectively reduced. Also, the quantification component serves as a mechanism that makes full use of the information, exploiting the redundant information to update the algorithm's parameters (see Section 6). The last component, dynamic adjustment of weights, enhances the capability of KRLS to track time-varying characteristics making the algorithm insensitive to outliers or mutations in the environments with noise or time-varying characteristics. The ALD criterion and the quantization follow the procedures described in Section 4 and Section 6 respectively. The coherence criterion [23] introduces the coherence coefficient μ:

$$\mu = \max_{j=1,2,\dots,i-1} |\kappa(\tilde{\boldsymbol{x}}_j), \boldsymbol{x}_i)| \le \mu_0 \tag{49}$$

where $\tilde{\boldsymbol{x}}_j$ is the j^{th} support vector in the existing dictionary until $i-1$ and μ_0 belongs to interval $[0, 1]$. If coefficient μ is no more than μ_0, then current input $\boldsymbol{x}_i$ is added to the dictionary. Regarding the dynamic adjustment of weights, this is a normalization extension of Eq. 27 leading to the following form:

$$\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i-1} + \frac{\tilde{\boldsymbol{K}}_i^{-1}\boldsymbol{q}_i(y_i - \tilde{\boldsymbol{k}}_{i-1}^T\boldsymbol{\alpha}_{i-1})}{\tau + \left\|\tilde{\boldsymbol{k}}_{i-1}^T\right\|^2} \tag{50}$$

where τ is a small parameter so that the denominator does not become zero.

Algorithm 9 ANS-QKRLS [31]

Input: $\{x_i, y_i\}$ *with* $i = 1 \dots N$

Initialization: choose kernel type κ, kernel parameter ξ and regularization parameter λ, $\tilde{\boldsymbol{K}}_1^{-1} = 1/(\lambda + \kappa(\boldsymbol{x}_1, \boldsymbol{x}_1))$, $\boldsymbol{\alpha}_1 = (\tilde{\boldsymbol{K}}_1^{-1} y_1)$, $\boldsymbol{P}_1 = 1$, center list $D_1 = \{\}$

for $i = 2 \dots N$ **do** ▷ Iterate over input-output pairs
  Compute kernel vector $\tilde{\boldsymbol{k}}_i$ by using Eq. 28
  $\tilde{\boldsymbol{a}}_i = \tilde{\boldsymbol{K}}_{i-1}^{-1}\tilde{\boldsymbol{k}}_i$
  $\delta_i = \lambda + \kappa(\boldsymbol{x}_i, \boldsymbol{x}_i) - \tilde{\boldsymbol{k}}_i^T\tilde{\boldsymbol{a}}_i$
  Compute distance between $\boldsymbol{x}_i$ and D_{i-1}: $dis(\boldsymbol{x}_i, D_{i-1}) = \min_{1 \le j \le size(D_{i-1})} \|\boldsymbol{x}_i - D_{i-1}^j\|$
  Compute coherence coefficient μ using Eq. 49
  if $\delta_i > \nu$, $\mu_i \le \mu_0$ **and** $dis(\boldsymbol{x}_i, D_{i-1}) > \varepsilon_{\mathbb{U}}$ **then**
    $D_i = \{D_{i-1}, \boldsymbol{x}_i\}$ ▷ Register $\boldsymbol{x}_i$ to dictionary
    $m_i = m_{i-1} + 1$ ▷ Update number of breaches
    Compute kernel matrix $\tilde{\boldsymbol{K}}_i^{-1}$ as in Eq. 19,

$$\boldsymbol{A}_i = \begin{bmatrix} \boldsymbol{A}_{i-1} & \boldsymbol{0} \\ \boldsymbol{0}^T & 1 \end{bmatrix}, \boldsymbol{P}_i = \begin{bmatrix} \boldsymbol{P}_{i-1} & \boldsymbol{0} \\ \boldsymbol{0}^T & 1 \end{bmatrix}$$

    Compute $\boldsymbol{\alpha}_i$ as in Eq. 20 ▷ Coefficient vector is calculated as in the standard KRLS
  else if $\delta_i \le \nu$ **or** $\mu_i > \mu_0$ **then**
    if $dis(\boldsymbol{x}_i, D_{i-1}) \le \varepsilon_{\mathbb{U}}$ **then**
      $D_i = D_{i-1}$, $\boldsymbol{q}_i = \frac{[\boldsymbol{K}_{i-1}^{j^*}]^{-1}\tilde{\boldsymbol{a}}_i^{j^*}}{1+(\tilde{\boldsymbol{a}}_i^{j^*})^T[\boldsymbol{K}_{i-1}^{j^*}]^{-1}\tilde{\boldsymbol{a}}_i^{j^*}}$, ▷ Dictionary unchanged
      $[\boldsymbol{K}_i]^{-1} = [\boldsymbol{K}_{i-1}]^{-1} - \boldsymbol{q}_i(\tilde{\boldsymbol{a}}_i^{j^*})^T[\boldsymbol{K}_{i-1}]^{-1}$,
      $\boldsymbol{\alpha}_i = \boldsymbol{\alpha}_{i-1} + \boldsymbol{q}_i(y_i - (\tilde{\boldsymbol{a}}_i^{j^*})^T\boldsymbol{\alpha}_{i-1})$ ▷ Quantize $\boldsymbol{x}_i$ to the closest center
    else
      $D_i = D_{i-1}$, $m_i = m_{i-1}$, ▷ Dictionary unchanged
      $\tilde{\boldsymbol{K}}_i = \tilde{\boldsymbol{K}}_{i-1}$, $\boldsymbol{A}_i = [\boldsymbol{A}_i^T, \tilde{\boldsymbol{a}}_i]^T$
      Compute $\boldsymbol{q}_i$ as in Eq. 25
      Compute $\boldsymbol{\alpha}_i$ as Eq. 50 ▷ Dynamic adjustment of coefficients using the normalized version of ALD Pass Path, i.e., normalized form of Eq. 27
    end if
  end if
end for

The pseudo-code of ANS-QKRLS is presented in Algorithm 9. Practically, the algorithm is divided into three operational cases. In the first case, if the ALD criterion breaches ($\delta_i > \nu$), the coherence coefficient μ_i of the current input is no more than than μ_0 and $dis(\boldsymbol{x}_i, D_{i-1}) > \varepsilon_{\mathbb{U}}$, then the input is added to the dictionary expanding its size. In the second case, if the ALD criterion holds or the current coherence coefficient μ_i is less than μ_0 but $dis(\boldsymbol{x}_i, D_{i-1}) < \varepsilon_{\mathbb{U}}$, then the dictionary remains unchanged and $\boldsymbol{x}_i$ is quantized to the closest center. Finally, if both of the above cases are not satisfied, then the current example can be expressed by the combinations of existing mutual independent components in the dictionary. Therefore, this input sample is discarded but the coefficient weights are adjusted accordingly.

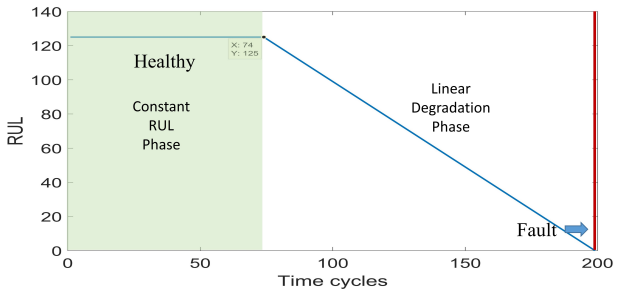


Figure 1: Piece-wise linear RUL function.

8. Remaining Useful Life Problem Description

Over the last decades, there has been an intensified research interest in the field of remaining useful life (RUL) prediction [38]. This term describes the progression of faults in Prognostics and Health Management (PHM) applications. In the fault detection concept, when a defect has occurred the objective is to identify it accurately from a list of potential failures. On the other hand, in the prognostics area, the objective is to predict the available time (life-cycle) before a failure occurs aiming to perform informed maintenance actions that minimize downtime and prevent critical failures.

The most widely studied dataset in the RUL domain is called C-MAPSS [46]. This dataset has been created by NASA simulating the degradation of aircraft engines (turbofan engines) utilizing the simulation tool Commercial Modular Aero-Propulsion System Simulation. A set of multivariate signals is provided including operational settings, temperature, pressure and mainly sensory data generated under open-loop system configurations, as depicted in Table 1. In this work, 17 of those measurements will be utilized, as indicated in green color. The C-MAPSS involves four sub-sets (FD001, FD002, FD003, FD004) that concern different operating conditions and fault modes. Each sub-dataset includes 21 sensory data and 3 operating settings among a training set and a testing set. These measurements practically start at a degradation level that is considered as healty and stop when failure is reached. In this work, evaluation tests will be performed using FD001 which includes 100 training and 100 testing units with 20631 and 13096 total instances across all engines respectively. Instances are distributed unevenly across the 100 engines, meaning that the behavior of each engine is represented by a different number of instances. This discrepancy is due to the varied number of operational cycles each engine undergoes from its healthy state to failure, reflecting the distinct operational life spans and conditions specific to each engine, thereby showcasing their unique degradation patterns. The objective is to predict the RUL of each turbofan engine at the end of its testing record. For RUL target label, a piece-wise linear degradation function is adopted limiting the RUL value at a maximum threshold. The maximum constant value of RUL is chosen as 125 time cycles and after that value, the engine starts to degrade at a certain point as illustrated in Fig. 1.

The utilization of a piece-wise linear model for predicting RUL reflects the degradation characteristics typical to turbofan engines. Initial stage of operation suggests a healthy period for the system, while degradation is increased towards its end-of-life phase. This justifies setting a piece-wise linear degradation model, aligning with findings from other works in the literature [57, 58, 59, 60], emphasizing the model's fidelity in mirroring the engines' life stages. Thus, a constant value of RUL is used at 125 cycles for healthy stage based on observed data. This approach mitigates the risk of RUL overestimation, crucial for ensuring reliable maintenance schedules and operational safety. Moreover, the linear degradation model represents the most natural choice in scenarios where specific prior knowledge of an appropriate degradation curve is lacking. By assuming a piece-wise linear degradation, the model offers a pragmatic balance between simplicity and the ability to reflect real-world engine behavior, particularly when detailed degradation patterns are unknown or hard to predict accurately.

8.1. Data Pre-processing

In the training stage, we consider two distinct cases where features are scaled using conventional normalization and standardization (z-score normalization) operations. For data normalization the input data are scaled to $[-1, 1]$,

whereas data standardization follows classic zero-mean and unit-variance. In both scaling cases, we distinct three sub-scenarios of handling data based on their sequence length during training phase. It is reminded that 100 engine cases constitute training data, but each turbofan engine includes different length of records. Thus, the three sub-scenarios are: a) natural handling of data with engines being unsorted; b) sorted engine instances based on their sequence length in a descending order; c) sorted engine instances by their sequence length in an ascending order; The latter two scenarios differ only in the order in which the algorithms will receive the input data (see Figure 2). The objective is to investigate whether the performance of KAF-based algorithms is affected when more importance is given during the training process to engines with either larger or smaller number of sequence length. It is important to evaluate the resiliency of KAF-based models under these cases that may lead them to local-minima, while also to study their behavior during the process of storing support vectors into their dictionaries. The idea of evaluating the performance of KAF-based algorithms under different sorted engine data emerged from Convolutional Neural Network (CNN) implementations that may follow sorting of training data based on engine sequence length in this application. Sorted data based on length of sequence data in CNNs is a way to reduce the amount of padding choosing a mini-batch size that divides the training dataset evenly. Therefore, we study the performance of KAF algorithms also under a recognized technique that enhances the training procedure of CNNs, although intuitively due to the process of the training mechanism in KAF-based algorithms seems like an extreme case that will degrade the prognostic performance.

8.2. Evaluation Metrics

The most widely evaluation metrics used in this problem are the root mean square error (*RMSE*) and an asymmetric scoring function (*Score*) [46]:

Table 1: C-MAPSS Dataset overview

	Symbol	Description details
1	op_setting_1	Operational setting 1 (environment variable)
2	op_setting_2	Operational setting 2 (environment variable)
3	op_setting_3	Operational setting 3 (environment variable)
4	T2	Total temperature at fan inlet
5	T24	Total temperature at LPC outlet
6	T30	Total temperature at HPC outlet
7	T50	Total temperature at LPT outlet
8	P2	Pressure at fan inlet
9	P15	Total pressure in bypass-duct
10	P30	Total pressure at HPC outlet
11	Nf	Physical fan speed
12	Nc	Physical core speed
13	Epr	Engine pressure ratio (P50/P2)
14	Ps30	Static pressure at HPC outlet
15	Phi	Ratio of fuel flow to Ps30
16	NRf	Corrected fan speed
17	NRc	Corrected core speed
18	BPR	Bypass Ratio
19	farB	Burner fuel-air ratio
20	htBleed	Bleed Enthalpy
21	Nf_dmd	Demanded fan speed
22	PCNfR_dmd	Demanded corrected fan speed
23	W31	HPT coolant bleed
24	W32	LPT coolant bleed

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} d_i^2} \tag{51}$$

$$Score = \begin{cases} \sum_{i=1}^{n} \exp\left(\frac{-d_i}{13}\right) - 1, & \text{for } d_i < 0 \\ \sum_{i=1}^{n} \exp\left(\frac{+d_i}{10}\right) - 1, & \text{for } d_i \geq 0 \end{cases} \tag{52}$$

where d_i stands for the prediction error with $d_i = RUL_{predicted} - RUL_{true}$ and n is the total number of instances. Positive prediction error means late prediction, while negative error is connected with early prediction. The *RMSE* induces equal penalization to both early and late predictions for the same absolute value of d_i. On the contrary, the asymmetric scoring function penalizes more late predictions introducing a more fair evaluation metric (see Fig. 3).

Indeed, from the maintenance point of view and in line with the risk-averse attitude in the aerospace industries, late forecasts do not allow maintenance to take place (thus more penalty applied), while very early forecasts may not be associated with major damage although they could waste maintenance resources (hence less penalization compared with late ones). Although, the scoring function is the most widely used evaluation metric in this application, there are some drawbacks that have been briefly identified [57]. First and foremost, much late predictions may dominate the performance of this metric. Also, there is lack of prognostic horizon enabling algorithms to be evaluated at a specific confidence level. Lastly, the scoring function favors those algorithms that artificially increase the performance (lowering *Score*) by underestimating the remaining useful life. For this reason, we utilize a supplementary metric to assess RUL predictions as {early}, {on time} or {late} as originally presented in [61] and later in [62, 63]. This metric is used in conjunction with *RMSE* and *Score* to provide an oversight and explainable view of the under-examination algorithms' behavior. Thus, this metric serves as a measurement of the state of prediction and this is given by:

$$State = \begin{cases} On\ time, & \text{for } -13 < d_i < 10 \\ Late, & \text{for } d_i \geq 10 \\ Early, & \text{for } d_i \leq -13 \end{cases} \tag{53}$$

It should be noted that someone could use more strict bounds or alternatively create more levels to discretize *State*. However, in this work we do not use solely this metric to evaluate the performance of KAF-based implementations. We use *State* as a supplementary metric to explain to some extent the performance difference between *RMSE* and *Score*, if any, while also to explain the behavior of KAF-based algorithms. This is an important feature that shows if an algorithm underestimates RUL (increased number of early predictions) or if late predictions increase dramatically the *Score* metric. Note also that even few but heavily late predictions may dominate the *Score* leading to a degraded performance. Figure 3 illustrates the levels of *State* as shaded areas for a given interval of prediction error, i.e. $[-50, 50]$.

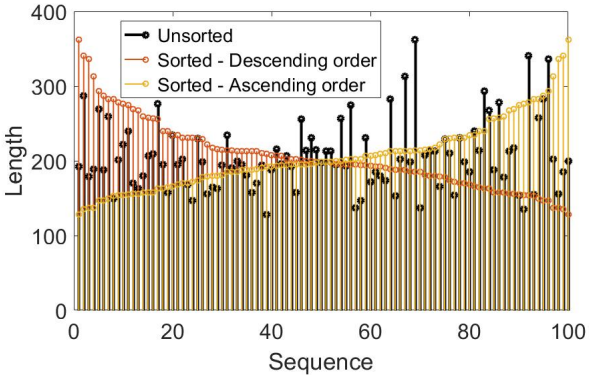


Figure 2: Three sub-scenarios of handling training data.

However, there are more evaluation metrics that have been reported in the literature. Prediction interval coverage probability (PICP) [64] measures whether the observed target RUL lies within the prediction interval with a probability $(1-\alpha)$ including lower and upper bounds of prediction interval for every test sample. NMPIW [64] is the normalized version of mean prediction interval width (MPIW) which gives a percentage of a range of expected RUL. PICP and NMPIW are conflicting, and a higher PICP will result in a wider NMPIW. The continuous ranked probability score (CRPS) and its weighted extension, CRPS ($CRPS^W$) [65], are metrics that evaluate the accuracy and sharpness of the estimated RUL distributions. The α-coverage and reliability score (RS) [65] evaluate the reliability of the RUL prognostics by quantifying overestimation and underestimation.

9. Evaluating KAF approaches in Remaining Useful Life: Intra-Comparison Analysis

In this work, we distinguish two evaluation cases based on feature scaling operations of normalization and standardization, as mentioned above. Three sub-scenarios stem under each case based on the method of treating the order of turbofan engines towards constructing the training data. More specifically, in both scaling cases we handle data either as they are given in the dataset (unsorted-natural order of engines) or we sort training examples by their sequence length in a descending and ascending order as described in 8.1. In all scenarios, 17 features are selected (colored features in Table 1). Regarding the selected KAF algorithms for comparison purposes, the most dominant sparsification candidate is chosen (ALD-KRLS), both strategies of growing and pruning (SW-KRLS and FB-KRLS), both quantification approaches (QKRLS and QKLMS) and lastly the two combined approaches that encapsulate sparsification and quantification (QALD-KRLS and ANS-QKRLS). For each algorithm, we vary the parameters that control the dictionary size. This approach was informed by the direct impact these parameters have on the size of the KAF networks and their overall performance. It is reminded that the term dictionary size typically refers to the number of basis functions (kernels) that are used in the model. For KAFs, each entry in the dictionary corresponds to a center of an RBF unit. The dictionary size therefore determines the complexity of the model: a larger dictionary can capture more complex functions but may also be more prone to overfitting and will require more computational resources to update and evaluate. Our evaluation explores these parameters, offering a comprehensive understanding of each algorithm's operational details and performance under various configurations. The performance of each KAF-based algorithm is evaluated using three metrics at different network sizes, with training time also considered as an additional indicative factor. This multi-metric assessment provides a holistic view of the algorithms' capabilities, necessary for understanding their behavior in practical RUL prediction scenarios. Our extensive evaluation, conducted on a conventional laptop with an AMD Ryzen 9 4900HS, 16GB RAM, and GeForce RTX 2060 Max-Q, aims not only to benchmark the current capabilities of KAF architectures in RUL prediction but also to guide future developments in the domain of predictive maintenance, particularly in scenarios where computational resources are a limiting factor.

9.1. Normalization Scaling Case

Table A.5 provides a comparison study for the case of normalization scaling and natural handling (unsorted data) of training engine data. The best performed network configurations under *Score*=650 are indicated in bold. The

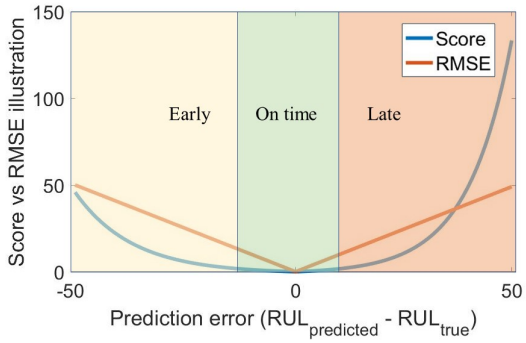


Figure 3: Visual representation of *Score*, *RMSE* and *State* (shaded areas) evaluation metrics.

budget-based approaches (SW-KRLS with $m = 1000$ and FB-KRLS with $m = 450$), as well as the ANS-QKRLS method with $m = 1140$ produced the best results. Between these three, FB-KRLS produced less {late} and {on time} predictions leading to an increased underestimation of RUL (37 {early} predictions). Note that QALD-KRLS balances between two sub-operations (ALD side and Quantization side) based on parameters configuration. Figure 4 illustrates the main *Score* results of Table A.5 against the network size for all algorithms. As presented, the algorithms ANS-QKRLS, QKLMS, ALD-KRLS and QALD-KRLS (ALD side) produce less oscillations as dictionary increases (network size). Moreover, the quantization approaches verify the property of reducing the numerical complexity as they present better results for smaller network size than the baseline ALD-KRLS algorithm. The fastest training time is observed in QKLMS as well as the ability of curbing the dictionary growth for the same testing performance.

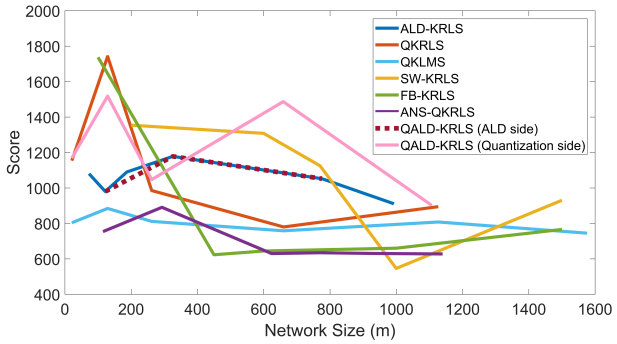


Figure 4: *Score* vs Network size for the main results provided in Table A.5 (normalization scaling with unsorted data handling).

Table A.6 summarizes the performance analysis of all aforementioned algorithms for similar network sizes as in Table A.5. The specificity here is that the training set has been composed sorting engines' order, based on their sequence length, in a descending (results in parentheses) and ascending manner (results in brackets). Take in mind that for the same amount of data samples considered during training phase, in the ascending sorted case more sub-sets of engine behaviors have been included (see Fig. 2). A general observation is that when descending sorting is adopted the algorithms tend to provide more {early} predictions, whereas when training data are sorted in ascending order more {late} predictions emerge. More specifically, the ALD-KRLS method slightly enhances its performance when using descending formulated data for training compared with unsorted case and it is strongly affected by ascending sorting case producing worst results. QKRLS seems to be negatively influenced by both sorting strategies compared with the unsorted case performance depicted in Table A.5. QKLMS outcomes a slightly enhancement with descending case while the opposite is observed for the ascending case. Both budget-based approaches (SW-KRLS, FB-KRLS) produce worse results with data sorting cases. Although, ANS-QKRLS algorithm presented worst results when using descending sorted training data, in ascending sorted data case produced a level of resiliency and slightly enhancement (with bigger network size) compared with standard natural (unsorted) data handling case. Finally, QALD-KRLS (ALD side) produced a slightly better performance when using descending sorted training data. The Quantization side of QALD-KRLS was negatively affected in both data sorting cases.

Regarding the behavior of KAF algorithms from the network size evolution perspective, Figure 5 illustrates the network growth curves for indicative configurations under different preprocessing scenarios (Unsorted, Sorted-Descending, Sorted-Ascending) within the normalization scaling case. As presented in Figure 5, irrespective of the KAF-based algorithm, in both sorting preprocessing operations less support vectors are added to the dictionary compared with the unsorted-natural operational case until 10000 training samples, then a "speed-up" is observed ending up in similar final network sizes. Also, around 6000 training samples an instant increase of network sizes is observed in the descending sorting preprocessing scenario. It should be reminded that at each iteration the number of samples investigated by algorithms is the same in all scenarios, but in the ascending sorting case more engines have been included as the number of iterations increases (see Fig. 2). Note that SW-KRLS and FB-KRLS algorithms have not been taken into account in Fig. 5 as their mechanism includes a pre-defined fixed dictionary size in which the oldest input sample and the least significant one is pruned respectively.

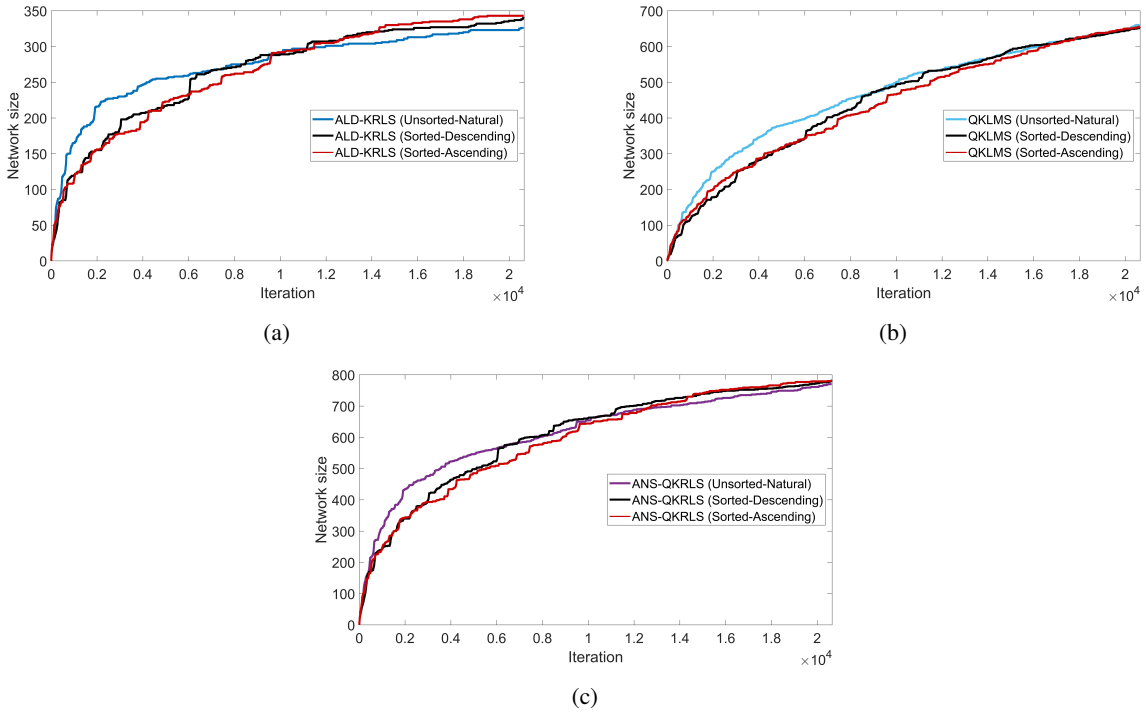


Figure 5: Network size evolution curves for the normalization scaling case. (a) ALD-KRLS with $m = 326$, while QALD-KRLS (ALD side) produces similar network growth behavior; (b) QKLMS and QKRLS with $m = 660$, while QALD-KRLS (Quantization side) follows similar behavior; (c) ANS-QKRLS with m=770.

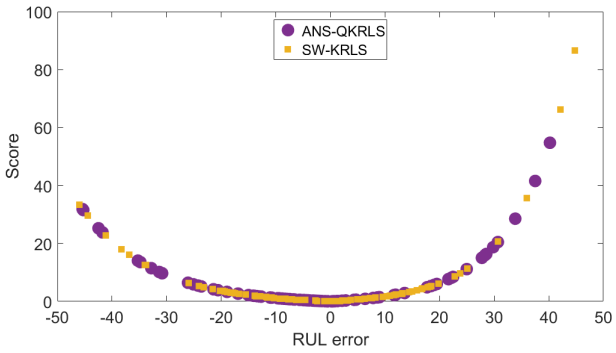


Figure 6: Concentration of individual prediction errors for the two best reported KAF algorithms in the normalization case. ANS-QKRLS (sorted training data in ascending order with 780 dictionary size and $Score = 537.79$) vs SW-KRLS (natural handling of training data with dictionary size of 1000 and $Score = 545.82$).

The best two performances are selected in order to inspect in more detail their performance outcomes. The first candidate for this comparison duel is SW-KRLS utilizing natural handling of training data with dictionary size of 1000 that produced $Score = 545.82$. The second approach is ANS-QKRLS using sorted training data in ascending order with network size of 780 and $Score = 537.79$. Figure 6 depicts in an unfolded way the results of both approaches providing insights about their performance from the *Score* evaluation perspective along 100 test engines. Although, ANS-QKRLS presents worse RMSE (18.36) than SW-KRLS case (RMSE=17.83), practically produces better concentration of individual predictions with four less {late} predictions. On the contrary, in SW-KRLS case there are 2 {late} predictions that contribute more than 150 to the total *Score* (the two upper right points contribute 152.85 to the

total of 545.82, i.e. around 28% increase in *Score* from these two {late} predictions).

9.2. Standardization Scaling Case

The same rationale as before is followed aiming to produce the performance of KAF-based algorithms under different parameter settings for each preprocessing scenario (Unsorted, Sorted-Descending, Sorted-Ascending). The main reason for adopting this scaling case, apart from an exploratory point of view, is to verify if KAF algorithms produce similar behavior regarding training data sorting scenarios as in the normalization scaling case. Table A.7 presents the KAF algorithms' performance for the conventional scenario of handling training data (Unsorted-natural). Again, the best *Score* under the level of 650 is given in bold. The best performed method is ANS-QKRLS utilizing $m = 160$ providing $Score = 468.58$. Also, this model provides less oscillations in performance for all network sizes sustaining dominant performance for all network size configurations.

Compared with Table A.5, ALD-KRLS algorithm provides similar results, QKRLS provides a slightly enhanced performance, QKLMS similar results, SW-KRLS and FB-KRLS produce worse results, ANS-QKRLS outcomes an increased performance and QALD-KRLS acts similarly. Table A.8 summarizes the performance of KAF-based approaches under the two sorting strategies. More specifically, ALD-KRLS provided an increased performance using descending sorting compared with unsorted scenario, while in the ascending scenario of handling training data no significant change is observed. QKRLS presented resiliency for some network sizes in both sorting scenarios, while also an enhanced performance is produced for small network size ($Score = 601.22$ and $Score = 633.32$ for $m = 22$ and $m = 23$ respectively). In the ascending scenario, QKLMS is strongly affected compared to the unsorted one, while in the descending scenario, an enhanced performance is observed for a small network size, achieving a ($Score = 556.1$ for $m = 22$). SW-KRLS and FB-KRLS are both influenced negatively by sorting scenarios, but mainly in the ascending one. ANS-QKRLS provides a degraded model in the descending scenario, while in the ascending case showed low level of resiliency for low network sizes but no better performance than the baseline (Unsorted-natural). Finally, QALD-KRLS follows similar behavior as ALD-KRLS and QKLMS for the corresponding sides (ALD side and Quantization side) respectively.

The best two models here that formulate the comparison duel are the ANS-QKRLS utilizing unsorted training data with dictionary size $m = 160$ producing $Score = 468.58$ and QKLMS using sorted data in descending order with dictionary size $m = 22$ leading to $Score = 556.1$. Figure 7 illustrates, in an analytical way, the contribution of each test engine prediction to the *Score* evaluation metric for the best reported models. QKLMS algorithm slightly promotes {early} predictions in respect to ANS-QKRLS, while the latter presents RUL error concentration in a narrower band around zero. Also, QKLMS is dominated by one severe {early} prediction (upper left point in Fig. 7) which corresponds to engine #45 for this algorithm.

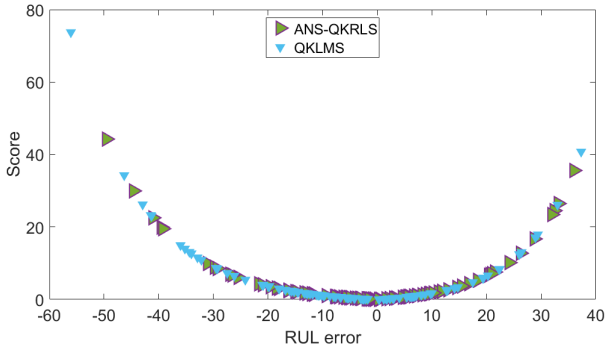
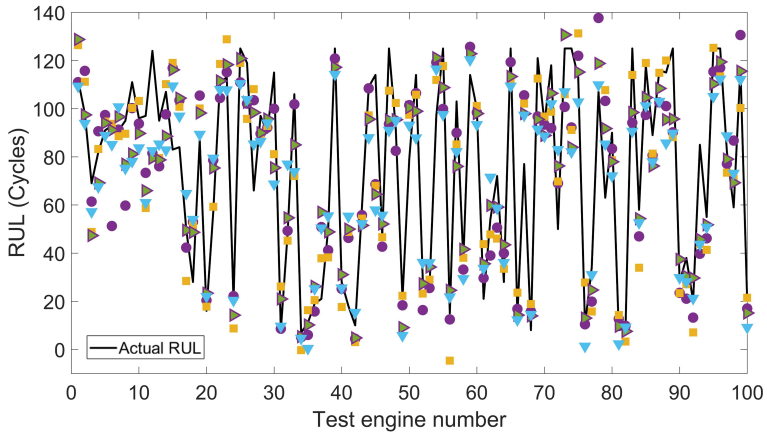


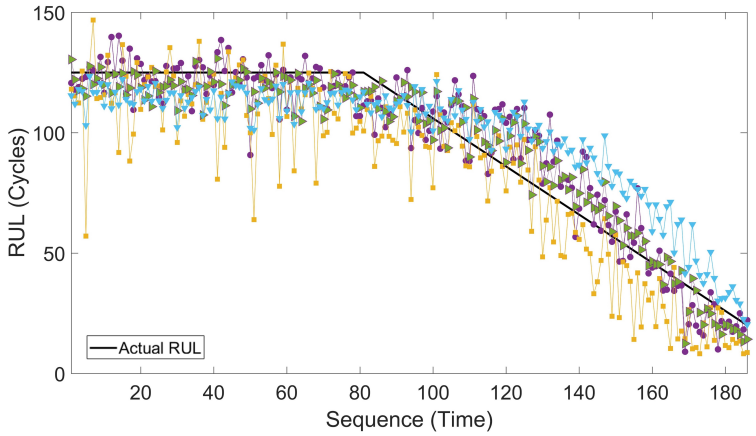
Figure 7: Concentration of individual prediction errors for the two best reported KAF algorithms in the standardization case. ANS-QKRLS (natural handling of training data with 160 dictionary size and $Score = 468.58$) vs QKLMS (sorted training data in descending order with dictionary size of 22 and $Score = 556.1$).

9.3. Performance Comparison within KAF Domain

The predicted RUL for each engine in the testing set, utilizing the best four KAF outcomes, is illustrated in Fig. 8 (a). An indicative comparative example between the predicted RUL values and the actual RUL values is depicted



(a)



(b)

Figure 8: Performance visualization for the best reported KAF approaches. (a) Predicted RUL for each testing engine against actual cycles; (b) Comparison between predicted and actual RUL for testing engine #24; Curve markers: ● ANS-QKRLS (normalization), ■ SW-KRLS (normalization), ▶ ANS-QKRLS (standardization), ▼ QKLMS (standardization)

in Fig. 8 (b). SW-KRLS method includes most fluctuating values during life cycle evolution compared with the other approaches, for this engine number. On the contrary, ANS-QKRLS (standardization) case presents the best tracking performance providing the most smooth curve. In all approaches, during early stage operation the prediction error is larger than in the late life phase of the engine. This is due to the fact that as the engine operates closer to failure, more degradation information becomes available, resulting in high late-stage prediction accuracy. Summarizing, Figure 8 (b) shows in higher level of detail the RUL tracking performance produced under testing engine #24, while Fig. 8 (a) depicts from an oversight perspective, unfolding the cumulative results of evaluation metrics, the overall performance for all testing engines.

9.4. Hyperparameter Tuning in KAF

Within the framework of KAF algorithms, a typical challenge is related with the proper selection of hyperparameters. One critical parameter is the bandwidth of Gaussian kernels, which significantly affects the smoothness of the model's output and its ability to accurately represent complex, non-linear trends in the data. Beyond these, each KAF

variant operates with intrinsic mechanisms that further influence the model's dictionary size—essentially the number of kernels or corresponding weights, given their RBF-like nature. This dictionary size, or network size, is directly impacted by the adjustment of specific hyperparameters, notably ν for algorithms leaning towards sparsity, and $\varepsilon_{\mathbb{U}}$ for those inclined towards a quantization approach. Indicatively, Figure 9 demonstrates the primary hyperparameter's effect on each algorithm under normalization scaling and natural handling conditions. Through these visual analysis, convergence towards lower *Score* values is observed as the dictionary size increases, underlining the influence of primal hyperparameters on the performance and complexity of KAF models.

In contexts where KAF algorithms are treated with a batch learning approach, techniques such as grid search, random search, and Bayesian optimization with cross-validation serve as standard methods for hyperparameter tuning. Among these, Gaussian process (GP) regression offers a more efficient and principled method, where optimal hyperparameters are identified by maximizing the log marginal likelihood. In particular maximum likelihood type-II (marginal likelihood maximisation or evidence maximisation) is a powerful generic way of adjusting hyperparameters via nonlinear optimization which scales linearly in the number of parameters. Adaptive methods, including adaptive kernel size [17], multikernel adaptive filtering [18], and Gaussian KAFs with adaptive kernel bandwidth [19], present tailored solutions in online learning scenarios. These techniques allow for real-time adjustments of the kernel size or bandwidth, ensuring the KAF model remains optimally tuned to the evolving data stream, thus enhancing both predictive performance and computational efficiency.

9.5. Impact of Preprocessing Operations on KAF Algorithms Performance

This section examines the influence of preprocessing operations on the performance of KAF algorithms in the context of RUL prediction. We explore the effects of two feature scaling methods, normalization and standardization, alongside the three distinct data handling strategies: unsorted (natural handling), descending, and ascending order sorting of training data. Performance metrics, such as *Score* distribution and the distribution of {early}, {on-time}, and {late} predictions, are evaluated to establish a comprehensive understanding of these preprocessing operations on the efficacy of KAF algorithms. More specifically, Figure 10 illustrates the spread and central tendency of final RUL prediction scores for each KAF algorithm, categorized by natural, descending order, and ascending order data handling methods, highlighting the influence of preprocessing strategies on algorithmic performance consistency. The stacked bar charts in Figure 11 present the proportional distribution of {early}, {on-time}, and {late} RUL predictions for various KAF algorithms under different data handling strategies.

9.5.1. Feature Scaling and Data Sorting Strategy Impact

Normalization and standardization preprocessing operations have been utilized to reshape the data landscape, impacting the performance trajectory of KAF algorithms. From Figure 10, it is evident that normalization contributes to a broader spread in final scores for algorithms like SW-KRLS and FB-KRLS. In contrast, ANS-QKRLS and ALD-KRLS exhibit a more condensed score distribution, which may indicate a more consistent performance across different dictionary sizes. The *Score* distribution unveils a more complex narrative: although a similar behavior between normalization and standardization can be observed in Figure 11, the impact of {late} predictions on the final score is more pronounced under normalization (Figure 10). This is manifested in the wider interquartile ranges and the presence of more outliers, indicative of certain late predictions being markedly later in the normalized scenario, thereby imposing a steeper penalty on the final *score*. Standardization generally compresses the score distribution, as observed for all algorithms, particularly in the context of natural handling. This finding suggests that standardization might be providing a more discriminate feature space, which is especially beneficial for these algorithms even when operating with smaller dictionary sizes. The stacked bar charts (Figure 11) do not exhibit a stark contrast in the proportion of {early}, {on-time}, and {late} predictions between normalization and standardization. However, it can be observed that descending sorting tends to provide more {early} predictions, whereas when training data are sorted in ascending order more {late} predictions emerge.

9.5.2. Algorithm-Specific Observations, Insights and Implications

The conducted analysis reveals algorithm-specific behaviors under different preprocessing regimes. ANS-QKRLS demonstrates an enhanced robustness under normalization across varied network sizes, suggesting an adaptive-related behavior in capturing degradation signals with minimal oscillation. SW-KRLS and FB-KRLS, while exhibiting a

wider variability in scores, could potentially benefit from tailored preprocessing operations to leverage their algorithmic strengths. ALD-KRLS and QALD-KRLS (ALD side) maintain a tighter score distribution, which might indicate

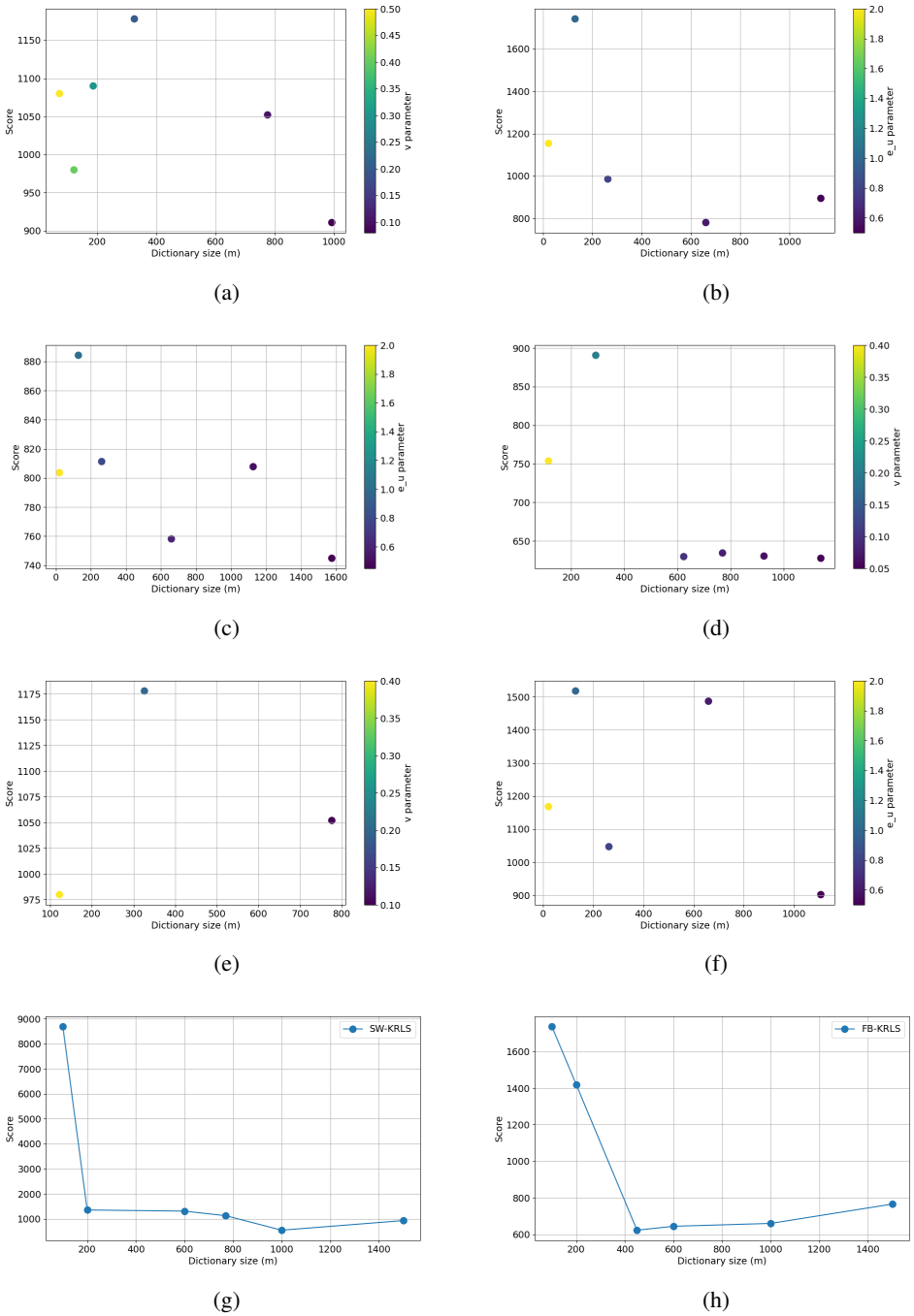


Figure 9: *Score* vs. Dictionary size under Natural handling - Normalization. (a) ALD-KRLS; (b) QKRLS; (c) QKLMS; (d) ANS-QKRLS; (e) QALD-KRLS (ALD side); (f) QALD-KRLS (Quantization side); (g) SW-KRLS; (h) FB-KRLS;

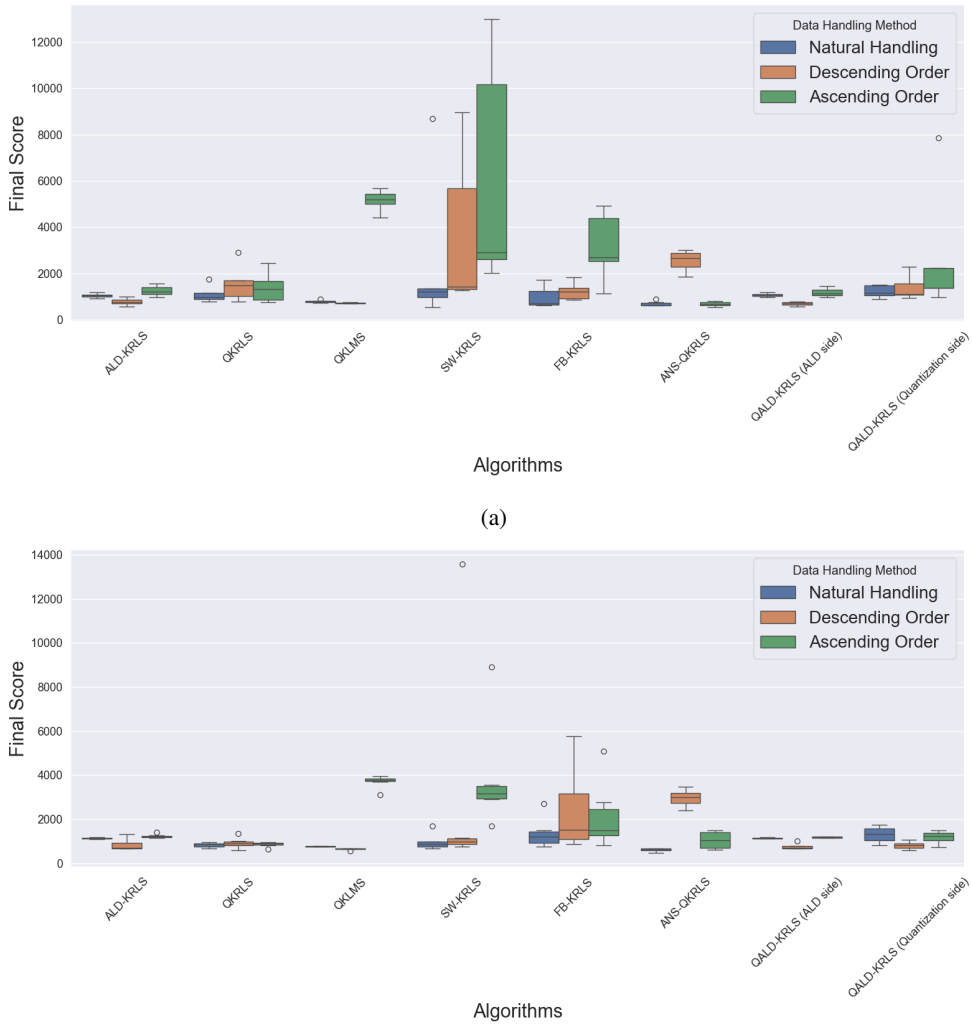
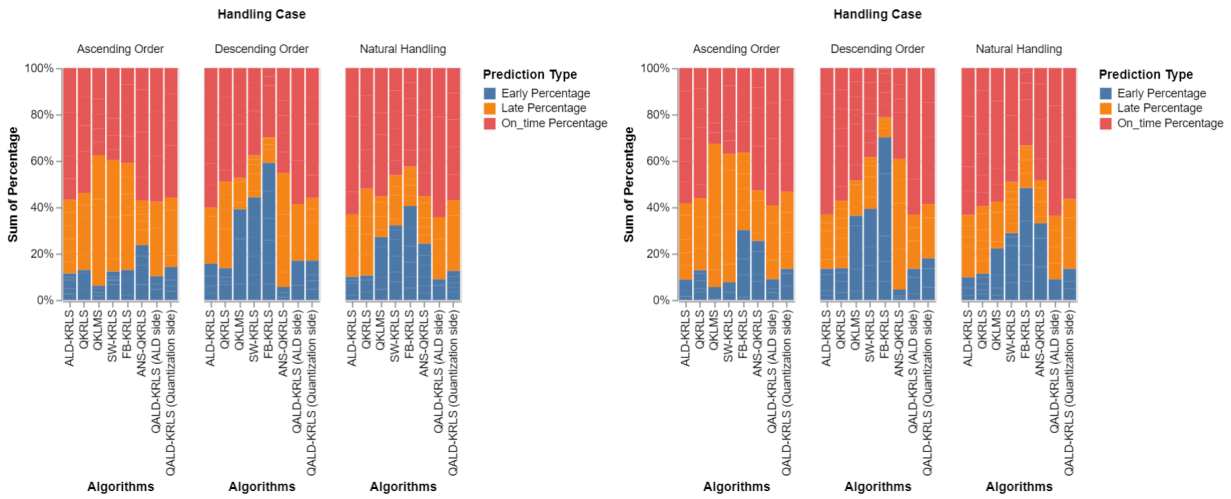


Figure 10: Boxplot distribution of final Scores across KAF algorithms. (a) Normalization; (b) Standardization;

a resilience to preprocessing variations and an inherent stability across different operational conditions.

The insights produced from this analysis are multifaceted. The choice of feature scaling method has a profound impact on the KAF algorithms' ability to generalize across different operational behaviors and to accurately detect degradation patterns. Normalization, while beneficial for stability, may lead to greater *Score* variability due to the magnified impact of late predictions. Standardization, conversely, reduces score dispersion, potentially offering a more consistent performance benchmark but requiring careful calibration to optimize early degradation detection capabilities.

Descending order sorting indeed starts with training data sequences that are longer, meaning a larger number of cycles before reaching the failure point. This approach effectively exposes algorithms to more extensive historical data right from the start, which can include stages closer to failure towards the end of each sequence but not necessarily in the initial stages of training. As training progresses with descending order sorting, algorithms are then exposed to shorter sequences. These shorter sequences, appearing later in the training process, represent engines with fewer cycles to failure from the outset of their data. This transition from longer to shorter sequences essentially means that, towards the later stages of training, the algorithms are operating with data that are closer to the failure events in terms



(a) (b)

Figure 11: Distribution of prediction timings for KAF algorithms. (a) Normalization; (b) Standardization;

of operational cycles remaining, which explains the increased number of {early} predictions. This approach contrasts with ascending order sorting, where algorithms start with shorter sequences, possibly making it harder initially to learn from more extended historical data since they begin with instances closer to failure, potentially making the {early} detection of degradation signs more challenging (more {late} predictions in this case overall).

In sum, preprocessing operations—feature scaling and data sorting—are not merely data manipulation techniques but pivotal factors that shape the very foundation of algorithmic performance in RUL prediction. They influence the algorithms' prediction timing, score variability, and in general their behavior. The insights from this study serve as a lessons learnt study regarding the importance of an informed selection of preprocessing techniques to harness the full potential of KAF algorithms in RUL, while also as an analysis to assess the performance of KAF algorithms with an increased difficulty level through data handling methods.

9.6. Training Time vs. Score Analysis

This analysis aims to unravel the efficiency and efficacy trade-offs inherent in these algorithms, from the perspective of training time (Figure 12), under various preprocessing schemes (natural, descending, and ascending orders coupled with normalization and standardization techniques). More specifically, ALD-KRLS showcases a balanced profile between accuracy and computational demand across preprocessing configurations. This algorithm maintains robust performance, particularly in scenarios where data normalization is applied, signifying its adeptness at handling standardized datasets with minimal compromise on training speed. QKRLS demonstrates a noteworthy proficiency in speed, but this comes at the cost of a slight dip in predictive accuracy. This trade-off is more pronounced under descending order preprocessing, suggesting that while QKRLS accelerates learning, it might miss finer details in rapidly changing data sequences. QKLMS emerges as the most time-efficient among the assessed algorithms, but with varying results. Its performance peaks under standardization procedures, indicating a preference for data with consistent variance, enabling faster convergence without substantial loss in accuracy. SW-KRLS and FB-KRLS both exhibit a steady increase in performance with increased training time, underscoring their suitability for applications where longer training periods are permissible for achieving higher accuracy. Specifically, SW-KRLS appears to leverage the sequential nature of data more effectively in descending order sorting, aligning with its inherent design. ANS-QKRLS presents an interesting dynamic where it excels in environments with ascending order preprocessing, suggesting an intrinsic capability to adapt to evolving data trends, which could be pivotal for real-time applications requiring timely updates. QALD-KRLS (ALD side) and QALD-KRLS (Quantization side) both illustrate distinct advantages in terms of precision and training duration. The ALD aspect enhances adaptability, making it well-suited for datasets exhibiting gradual degradation patterns, whereas the Quantization facet emphasizes speed, ideally servicing scenarios with

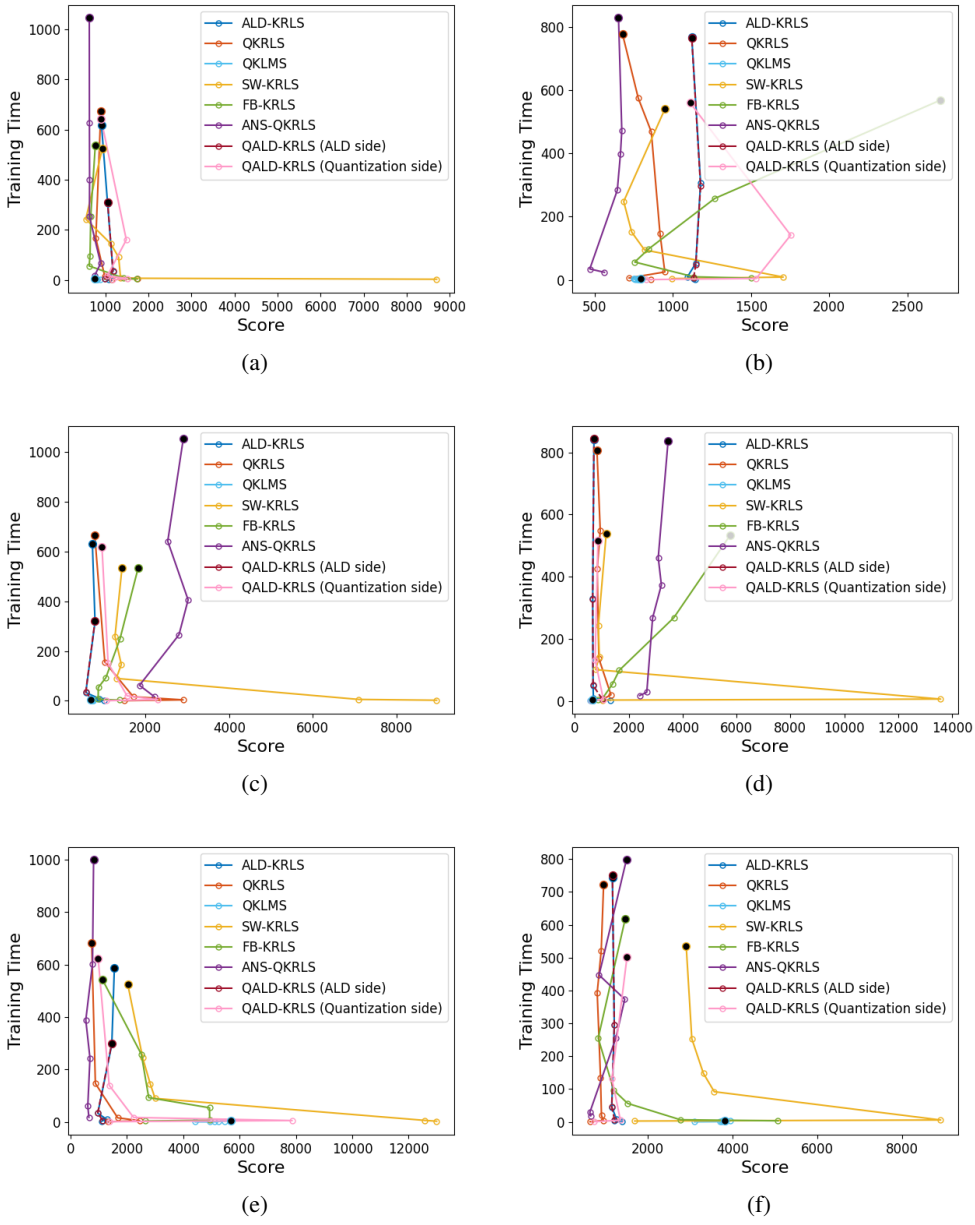


Figure 12: Training time vs. *Score* comparison for KAF algorithms. (a) Natural handling - Normalization; (b) Natural handling - Standardization; (c) Descending - Normalization; (d) Descending - Standardization; (e) Ascending - Normalization; (f) Ascending - Standardization;

stringent time constraints.

The insights from this analysis underscore the balance between training time and prediction accuracy across different KAF algorithms and preprocessing methods. The observed trends advocate for a tailored approach in selecting the appropriate KAF algorithm based on the specific requirements of the task at hand, such as urgency (reflected in training time) versus precision (mirrored in the *Score*). Furthermore, this examination aids in establishing a more informed framework for future research and practical applications, especially in areas demanding quick and reliable

predictions, like Remaining Useful Life (RUL) estimation.

9.7. Advantages and Limitations of KAF Algorithms in RUL Prediction

An extensive evaluation study has been conducted regarding the predicting properties and learning capabilities of KAF-based algorithms in a well-known RUL application. Seven candidates have been selected for intra-comparison purposes within KAF universe in terms of performance, behavior, training time, smoothness and reliability. Also, the impact of different feature scaling scenarios and diverse preprocessing methods has been assessed, presenting ANS-QKRLS as the most resilient approach.

Generally, online learning algorithms operate adapting their weights, in real-time, per training sample. This leads to requirements regarding computational and memory resources, especially in large time series applications. While KRLS is a straightforward extension of the RLS method, it faces scalability issues due to its computational and memory burden, which scale linearly with the dataset size. To address these challenges, the ALD-KRLS variant enhances KRLS by introducing a sparsification mechanism that limits the growth of the dictionary. Several research efforts led to sparsification, quantization, dictionary budget and combined approaches, as presented in previous sections, overcoming scaling limitations.

For instance, quantized algorithms streamline parameter count and training duration at a slight accuracy cost. Budget-based methods behave similarly, and have the advantage that they may be suited for tracking changes in the model underlying the observed data. Notably, SW-KRLS has demonstrated strong performance in this context, though its reliance on a fixed dictionary size increases its sensitivity to data preprocessing techniques. In general, ALD-KRLS outperforms quantized variants by optimizing a square cost function directly on input data, leading to more judicious center selection. However, quantized versions still deliver adequate outcomes with reduced training time. The hybrid models, QALD-KRLS and ANS-QKRLS, amalgamate ALD-KRLS's efficient center selection with the reduced parameterization of quantized methods, offering an improved balance between computational efficiency and performance. A summary table (Table 2) and the following discussion outline the advantages, limitations, and situational suitability of each KAF algorithm:

- **ALD-KRLS** is recognized for its stability and adaptability across diverse operational conditions, effectively capturing degradation patterns in RUL prediction tasks. Despite its robustness, ALD-KRLS may not always achieve the highest performance levels compared to other algorithms and requires more computational resources, which could be restrictive in time-sensitive or resource-limited settings. This algorithm is ideally suited for scenarios where consistent and reliable performance is prioritized over computational efficiency, particularly in complex environments where accurate degradation tracking is crucial.

- **QKRLS** offers good approximation abilities with reduced computational complexity through its quantization feature, which controls dictionary size growth. However, quantization can lead to a loss of information, impacting the precision and reliability of predictions. While QKRLS provides some computational advantages, its operational complexity is higher than that of QKLMS, as its computational demands scale quadratically with the dictionary size. This characteristic makes it less ideal for extremely resource-limited applications. QKRLS is suitable for applications that can handle its computational requirements.

- **QKLMS** excels in computational efficiency and simplicity of implementation, characterized by rapid update rules and no need for matrix inversion, making it highly scalable and suitable for non-stationary environments and incremental learning. However, it tends to exhibit lower prediction accuracy and is more sensitive to outliers compared to more complex KAF models. Furthermore, its quantization process can compromise data integrity, especially in noisy settings. QKLMS is most effective in large-scale systems requiring swift processing and real-time updates, where its operational speed and efficiency are crucial advantages.

- **SW-KRLS** effectively adapts to rapid changes by using a sliding-window mechanism that prioritizes recent data, making it ideal for environments with time-varying data models. This focus on new information, however, can reduce consistency and obscure long-term trends, potentially compromising overall accuracy. SW-KRLS is best suited for applications with limited memory where immediate responsiveness is more critical than historical precision.

- **FB-KRLS** maintains effective dictionary management with a fixed size, ensuring stable computational demands and optimizing vector selection over time, unlike the potentially still unbounded dictionary growth in ALD-KRLS. Its fixed-budget nature, however, may limit flexibility in adapting to new and diverse data trends. FB-KRLS is best suited for stable environments where computational predictability and maintaining performance with known overheads are crucial, without the need to handle highly dynamic data shifts.
- **ANS-QKRLS** balances computational efficiency with prediction accuracy and is equipped with sparsification and quantization techniques suitable for time-varying environments. It maintains stable performance across various network sizes and adapts effectively to new data, making it practical for RUL prediction tasks. The algorithm keeps a manageable dictionary size, optimizing efficiency without significantly impacting accuracy. However, ANS-QKRLS requires careful tuning of its adaptive techniques to maintain performance. It is well-suited for environments with changing operational conditions and complex degradation patterns, where stability and adaptability are necessary.
- **QALD-KRLS** combines quantization and ALD to balance efficiency with accuracy, suitable for dynamic environments with variable data. However, tuning complexity and potential information loss from quantization may impact precision. This model is ideal for large-scale systems where adaptability must align with limited computational resources.

Table 2: Summary of advantages, limitations, and contextual fit for each KAF algorithm

Algorithm	**Advantages**	**Limitations**	**Contextual Fit**
ALD-KRLS	Robustness, adaptability	Computational demand	Complex environmnets needing reliability
QKRLS	Good approximation, low computational complexity	Higher computational demand than QKLMS, potential information loss	Suited for systems that can manage higher computational needs
QKLMS	Computational efficiency, scalability	May produce lower accuracy, outlier sensitivity	Effective in large-scale systems requiring fast processing
SW-KRLS	Adapts quickly to changes, prioritizes recent data	May obscure long-term trends, less historical precision	Best for environments needing immediate responsiveness with limited memory
FB-KRLS	Stable computational demands, optimized vector selection	Limited flexibility in adapting to new data trends	Ideal for stable environments where predictability is crucial
ANS-QKRLS	Robustness, balance between accuracy and computational efficiency	Requires careful tuning	Varied operational conditions, complex degradation patterns
QALD-KRLS	Robustness, computational efficiency	Tuning complexity, potential information loss	Dynamic scenarios, Efficiency-required scenarios

10. Comparisons with Other Approaches and Discussion: Inter-Comparison Analysis

An inter-comparison analysis is needed to rank KAF performance with other machine learning (ML) and deep learning (DL) approaches reported in literature. For this reason, Table 3 summarizes KAF-based and other, mostly network-based approaches, providing a comprehensive performance report in terms of *RMSE* and *Score* evaluation metrics. It is evident from Table 3, that KAFs outperform more than half of the neural network models in terms of *Score*. The average *Score* of the 12 KAF-based approaches presented in Table 3 is around 579. Especially, ANS-QKRLS algorithm is ranked as 21^{st} among 58 neural network approaches in terms of *Score* metric. It should be underlined that in this work, a plain window size of just 1 is used (no time window), while most approaches in the literature reported in Table 3 use a sliding window with length equal to 30. Moreover, Fig. 13 illustrates a visualization mapping of different methods applied in RUL problem across *Score* metric. Note that the training time is not included in Table 3 due to the different simulation equipment adopted in each literature reported case. Moreover, most approaches reported in literature do not provide information regarding training time. Indicatively, the best performed KAF algorithm (ANS-QKRLS with network size 160) used 34.53 seconds for training utilizing a conventional laptop

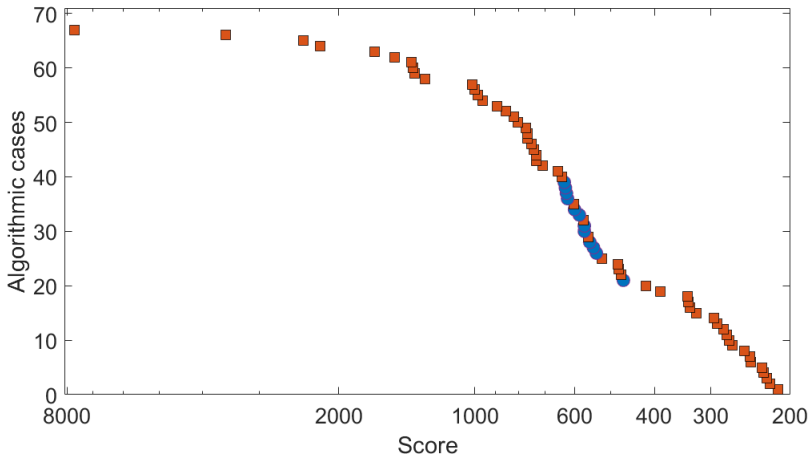


Figure 13: Visualization of Table 3 in terms of *Score* in logarithmic scale. ● KAF family ■ ML and DL approaches

with AMD Ryzen 9 4900HS and 16GB RAM. At the same time, AGCNN [66], one of the most dominant models in terms of *Score*, used 475.47 seconds for training, in the same C-MAPSS sub-dataset (FD001), utilizing a much better desktop simulation machine with Intel Xeon W-2155 CPU and 64GB RAM. Indeed, the performance difference between these two models exists, while also complexity resulting in training time gap difference. Other approaches adopt more exhaustive training mechanisms that apply genetic algorithm to tune the diverse amount of hyper-parameters resulting in 60 hours of training time [67]. KAF architectures follow a simple RBF network topology, while also they are suitable for online applications.

Compared with deep learning, kernel-based methods do not utilize a large number of weight parameters. Their trainable parameters are mainly related with the dictionary length m, i.e., the network size reported in Table 3. Note that, in the context of KAFs the dictionary size means the number of kernel centers or basis functions that are actively used in the model. In RBF-like networks, each of these centers corresponds to the center of an RBF kernel. At the same time, the coefficients or weights are multiplied by the kernel functions' outputs before being summed to form the final output of the KAF. There is typically one weight per kernel function, so while "dictionary size" technically refers to the number of kernels, it also dictates the number of weights since each kernel will have one associated weight. In deep neural network implementations, the trainable parameters typically range from thousands to even millions of weights for prediction purposes. For this reason, we illustrate indicatively in Table 4 the number of trainable parameters for a set of neural network-based approaches applied in FD001 set of C-MAPSS. Also, a qualitative comparison in terms of the number of trainable weights is depicted in Fig. 14, in logarithmic scale, serving as a visualized form of Table 4. Note that, if Fig. 14 were in linear scale with points being proportional to the size of trainable weights, then a few of the orange rectangles would be larger than the page itself. Indeed, the number of trainable weights in KAF approaches is significantly lower. In KAF-related algorithms, higher training throughput can be obtained regarding online learning problems due to the lower number of trainable weights.

It is important for future KAF implementations to exploit more weights in providing an enhanced performance, as a consequence of a more balanced trade-off between the competing objectives of computational burden and performance. Before examining the potential future directions for KAF architectures, it is pertinent to highlight their key advantages over neural network approaches, which are fundamental to understanding their role in prognostic engineering. The merits of KAF in comparison to neural networks are encapsulated in several key aspects:

1. Computational Simplicity: KAFs typically require fewer parameters, simplifying model complexity and computational demands.
2. Adaptability in Online Learning: KAFs are inherently suited for online learning environments, providing efficient real-time data processing and model adaptability.
3. Efficient Nonlinear Modeling: The kernel trick in KAFs facilitates effective handling of nonlinear relationships without the need for deep or complex architectures.

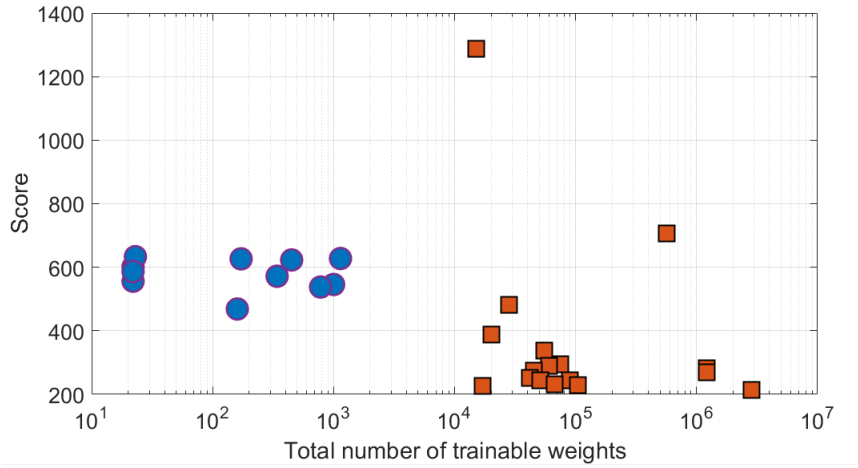


Figure 14: Qualitative representation of total number of trainable parameters versus *Score* in logarithmic scale. ● KAF family ■ DL approaches

4. Reduced Overfitting Risks: The lower number of parameters in KAFs can lead to less overfitting, particularly in limited data scenarios.
5. Efficient Use of Training Data: KAFs can often generalize effectively from smaller datasets, an advantage in cases where data collection is challenging or limited.
6. Rapid Training and Robustness: KAFs offer quicker training times and robustness to data variability, crucial in non-stationary environments.

These advantages not only underscore the potential of KAFs in applications where computational efficiency and adaptability are crucial but also open exciting avenues for future research.

However, while the comparative efficiency and simplicity of KAF architectures are apparent, it is critical to identify inherent challenges and limitations when juxtaposed against more complex neural network models. The simplicity of KAFs, primarily reflected through their straightforward RBF network topology and lower number of trainable weights, while advantageous for online applications and computational demands, may also constrain the depth of data representation and the produced performance compared to deep learning approaches. This limitation is particularly pronounced in scenarios requiring the capture of detailed, multi-dimensional relationships within large datasets, where neural networks, with their extensive trainable parameters, can produce higher performance results. While KAFs can scale to a degree by increasing their dictionary size, this scalability doesn't fully bridge the substantial gap in model complexity when compared to neural networks. Essentially, even as KAFs grow in size and capability, they may still fall short of achieving the same level of detailed data representation and processing power that is inherent to neural network models, particularly those with deep architectures and vast numbers of trainable parameters. Furthermore, the training mechanisms of KAFs, though generally less resource-intensive, might not afford the same level of optimization granularity provided by the extensive hyperparameter tuning and deep architectural configurations characteristic of advanced neural network models. Therefore, the merits of KAF models in conjunction with the decent produced performance may be further exploited in hybrid implementations with neural networks and the exploration of deep-learning-inspired hierarchical KAF structures arises. Such approaches could leverage the strengths of both KAFs and neural networks, potentially leading to more sophisticated and efficient prognostic tools. For instance, the conceptual idea of reorganizing the typical single-layer kernel-based model into a deep hierarchical structure has been implemented in [87] proposing deep kernel recursive least squares for two- and three-dimensional problems. In summary, the evaluation results pave the way for establishing KAF-based implementations in prognostic engineering applications, promoting an intensified interest in formulating KAF algorithms in hybrid structures or exploring deep learning-inspired KAF architectures. These developments could enhance the performance of KAFs while maintaining their advantageous trade-off between computational efficiency and prediction accuracy, potentially leading to more dominant models in the prognostics area.

11. Conclusion

This study has presented a comprehensive evaluation of Kernel Adaptive Filtering (KAF) algorithms in the context of Remaining Useful Life (RUL) prediction for aircraft engines, juxtaposed with an extensive range of neural network approaches, encompassing around 60 different models. The experiments are performed on the well-known, and publicly available, C-MAPSS dataset. Our findings reveal that KAF algorithms outperform more than half of the neural network models reported in the literature, with ANS-QKRLS outperforming two-thirds of those models in terms of *Score* metric. However, it is crucial to acknowledge the inherent limitations associated with KAFs, particularly when faced with the requirement to capture complex, multidimensional data relationships, a domain where deep learning models often exhibit superior performance due to their extensive parameter sets and deep architectures. Although, KAF architectures may not surpass the most advanced neural networks in performance metrics, they demonstrate decent prognostic accuracy that is coupled with important merits in terms of computational efficiency and training time. The evaluation of seven KAF variants highlighted the resilience and adaptability of these algorithms, with ANS-QKRLS emerging as a notably robust approach within the KAF family. The study's comparative analysis underscores the trade-offs between computational burden and predictive accuracy, showcasing KAFs as a viable option in applications where model simplicity and rapid training are advantageous.

Our research enriches the field with a detailed analysis of the operational spectrum of KAF algorithms, shedding light on their efficacy and applicability. The inherent characteristic of KAFs, having fewer trainable parameters, in conjunction with their commendable performance, renders them particularly suitable for prediction contexts where computational resources are constrained. Looking ahead, the results from this study pave the way for future research into hybrid models that blend the strengths of KAFs with the depth and complexity of neural networks. Such explorations could potentially lead to the development of more sophisticated prognostic tools, enhancing performance while maintaining computational efficiency. The possibility of deep-learning-inspired hierarchical KAF structures presents an intriguing avenue for further investigation, promising advancements in the field of prognostic engineering. In conclusion, this study not only benchmarks the current state of KAF architectures in RUL prediction but also opens up new horizons for their application in prognostic engineering, advocating for their consideration in hybrid and advanced algorithmic structures. The balance between computational efficiency and prediction accuracy that KAFs offer is likely to make them an increasingly relevant choice in the evolving landscape of machine learning applications in mechanical systems.

Appendix A. Comparative Results of Feature Scaling Techniques and Engine Data Ordering by Sequence Length Across Various KAF Algorithms

Table 3: Comparison of best performed KAF-based algorithms with other approaches reported in the literature regarding FD001 dataset.

KAF-based approaches						
Algorithm	Scaling mode	Data handling	Network size	TW	RMSE	Score
SW-KRLS	Normalization	Unsorted-Natural	1000	1	17.83	545.82
FB-KRLS	Normalization	Unsorted-Natural	450	1	19.14	622.99
ANS-QKRLS	Normalization	Unsorted-Natural	1140	1	18.62	627.55
ALD-KRLS	Normalization	Sorted-Descending	341	1	17.18	571.81
ANS-QKRLS	Normalization	Sorted-Ascending	780	1	18.36	537.79
QALD-KRLS	Normalization	Sorted-Descending	341	1	17.18	571.72
ANS-QKRLS	Standardization	Unsorted-Natural	160	1	17.58	**468.58**
QKRLS	Standardization	Sorted-Descending	22	1	19.02	601.22
QKLMS	Standardization	Sorted-Descending	22	1	19.28	556.1
QKRLS	Standardization	Sorted-Ascending	23	1	18.95	633.32
ANS-QKRLS	Standardization	Sorted-Ascending	172	1	18.22	626.45
QALD-KRLS	Standardization	Sorted-Descending	22	1	18.23	586.5

Machine learning and Deep learning approaches reported in literature			
Algorithm	TW	RMSE	Score
SVM [68]	30	40.72	7703.33
Echo State Network with Kalman Filter [69]	-	63.46	-
DW-RNN [70]	20	22.52	-
ESN-FCN [71]	1	21.67	3555
SVR [72]	1	21.74	2394
LR [72]	1	23.45	2200
MTL-RNN [70]	20	21.47	-
ETR [68]	30	23.76	1667.86
RVR [57]	1	23.79	1502.9
SVR [57]	1	20.96	1381.5
CNN [72]	-	19.7	1372
ETR [68]	1	22.05	1359.38
First CNN attempt [57]	15	18.45	1286.7
AE-FCN [71]	1	19.28	1014
DBN [68]	1	18.48	1001.44
LSTM [72]	-	18.98	983
MLP [68]	1	18.48	959.63
LASSO [68]	1	22.43	894.21
SVM [68]	1	20.58	852.07
CNN+RNN [73]	31	16.89	820.67
Random Forest [68]	1	20.23	802.23
Multi-Stage-RUL GB [72]	-	17.92	772
Multi-Stage-RUL SVM [72]	-	17.12	765
SKF [68]	1	19.24	762.85
Multi-Stage-RUL LSTM [72]	-	17.26	748
Extreme Learning Machine [68]	1	19.40	740.52
Multi-Stage-RUL CNN [72]	-	16.89	732
KNR [68]	30	20.46	729.32
CNN-FCN [71]	1	16.35	706
LASSO [68]	30	19.74	653.85
MODBNE [68]	1	17.96	640.27
KNR [68]	1	19.73	604.26
GB [68]	1	18.80	575.04
MLP [68]	30	16.78	560.59
Extreme Learning Machine [68]	30	17.27	523
LSTMBS [74]	31	14.89	481.1
Random Forest [68]	30	17.91	479.75
GB [68]	30	15.67	474.01
DBN [68]	30	15.21	417.59
2-layer LSTM [75]	31	16.74	388.68
Deep LSTM [76]	-	16.14	338
Trend attention Fully Convolutional Network [77]	31	13.99	336
MODBNE [68]	30	15.04	334.23
Attention-Based LSTM [78]	30	14.53	322.44
BiLSTM [79]	30	13.65	295
CNN-LSTM [80]	32	14.40	290
GHDR-FL [81]	30	11.58	281.65
CapsNet [82]	30	12.58	276.34
DCNN [58]	30	12.61	273.7
GHDR+LSTM+FC [81]	30	11.45	268.72
LSTM-MLSA [83]	-	11.57	252.86
Embedded Attention-based Parallel Network-ResLSTMa [84]	40	12.11	245.32
HDNN [60]	30	13.02	245
LSTM-RBM [67]	-	12.56	231
Distributed Attention-Based TCN [59]	40	11.78	229.48
AGCNN [66]	30	12.42	225.51
HAGCN [85]	30	11.93	222.3
BiGRU-TSAM [86]	30	12.56	**213.35**

Table 4: Total number of trainable parameters for a sub-set of DL approaches reported in Table 3.

Neural Network approaches reported in literature	Trainable weights
First CNN attempt [57]	≈ 15,000
CNN-FCN [71]	≈ 566,000
LSTMBS [74]	≈ 28,000
2-layer LSTM [75]	≈ 20,000
Deep LSTM [76]	≈ 55,000
BiLSTM [79]	≈ 75,000
CNN-LSTM [80]	≈ 60,000
GHDR-FL [81]	≈ 1,205,000
DCNN [58]	≈ 45,000
GHDR+LSTM+FC [81]	≈ 1,208,000
LSTM-MLSA [83]	≈ 42,000
Embedded Attention-based Parallel Network-ResLSTMa [84]	≈ 90,000
HDNN [60]	≈ 51,000
LSTM-RBM [67]	≈ 67,000
Distributed Attention-Based TCN [59]	≈ 105,000
AGCNN [66]	≈ 17,000
BiGRU-TSAM [86]	≈ 2,825,000

Table A.5: Performance comparison of the KAF-based algorithms using normalization scaling method and natural handling of engine order in the training phase.

Algorithm	h	λ	ν	η	$\varepsilon_{\mathbb{U}}$	μ_0	τ	m	RMSE	Score	Early - On time - Late	Training Time (sec)
ALD-KRLS	1	0.01	0.5	-	-	-	-	73	18.56	1080	13-59-28	2.29
	1	0.01	0.4	-	-	-	-	122	17.32	979.89	10-65-25	4.46
	1	0.01	0.3	-	-	-	-	187	17.36	1090	10-65-25	9.36
	1	0.01	0.2	-	-	-	-	326	17.73	1178	9-65-26	33.79
	1	0.01	0.1	-	-	-	-	776	17.6	1052	8-63-29	304.78
	1	0.01	0.08	-	-	-	-	993	17.34	910.7	10-62-28	616.84
QKRLS	1	0.01	-	-	2	-	-	21	20.89	1154	18-38-44	0.64
	1	0.01	-	-	1	-	-	129	21.08	1742	8-47-45	4.75
	1	0.01	-	-	0.8	-	-	262	18.78	985.43	9-53-38	14.53
	1	0.01	-	-	0.6	-	-	660	17.62	779.83	8-62-30	165.6
	1	0.01	-	-	0.5	-	-	1127	17.43	894.3	10-60-30	672.35
QKLMS	1	-	-	0.01	2	-	-	21	19.85	803.67	28-53-19	0.35
	1	-	-	0.01	1	-	-	129	19.81	884.27	30-53-17	0.84
	1	-	-	0.01	0.8	-	-	262	18.86	811.3	29-53-18	1.19
	1	-	-	0.01	0.6	-	-	660	18.6	758.19	25-57-18	2.1
	1	-	-	0.01	0.5	-	-	1127	18.59	807.77	26-58-16	2.87
	1	-	-	0.01	0.45	-	-	1576	18.58	744.9	25-58-17	3.63
SW-KRLS	1	0.1	-	-	-	-	-	100	31.49	8693	47-37-16	2.44
	1	0.1	-	-	-	-	-	200	22.64	1354	25-41-34	6.6
	1	0.1	-	-	-	-	-	600	23.88	1308	42-44-14	90.3
	1	0.1	-	-	-	-	-	770	23.06	1125	43-43-14	144.5
	1	0.1	-	-	-	-	-	1000	17.83	**545.82**	23-54-23	242.8
	1	0.1	-	-	-	-	-	1500	17.02	930.19	13-58-29	525.1
FB-KRLS	1	0.1	-	0.00001	-	-	-	100	21.79	1737	31-43-26	3.24
	1	0.1	-	0.00001	-	-	-	200	22.34	1420	30-42-28	7.33
	1	0.1	-	0.00001	-	-	-	450	19.14	**622.99**	37-47-16	55.63
	1	0.1	-	0.00001	-	-	-	600	20.51	644.22	41-45-14	95.8
	1	0.1	-	0.00001	-	-	-	1000	21.71	660.09	49-41-10	252.5
	1	0.1	-	0.00001	-	-	-	1500	23.5	765.83	55-37-8	536
ANS-QKRLS	1	0	0.4	-	0.05	0.95	0.00001	115	17.71	753.62	17-61-22	17.94
	1	0	0.2	-	0.05	0.95	0.00001	293	18.25	890.71	19-60-21	65.83
	1	0	0.1	-	0.05	0.95	0.00001	624	18.62	629.68	25-51-24	253.92
	1	0	0.08	-	0.05	0.95	0.00001	770	18.9	634.4	28-53-19	400.88
	1	0	0.065	-	0.05	0.95	0.00001	926	18.78	630.38	26-55-19	628.33
	1	0	0.05	-	0.05	0.95	0.00001	1140	18.98	**627.55**	31-52-17	1046.1
QALD-KRLS	1	0.01	0.4	-	0.1	-	-	122	17.32	979.89	10-65-25	5.41
	1	0.01	0.2	-	0.1	-	-	326	17.73	1178	9-65-26	35.73
	1	0.01	0.1	-	0.1	-	-	776	17.6	1052	8-63-29	309.77
	1	0.01	0.04	-	2	-	-	21	19.86	1168	17-55-28	0.95
	1	0.01	0.04	-	1	-	-	129	19.46	1518	10-60-30	4.82
	1	0.01	0.04	-	0.8	-	-	262	18.5	1047	11-59-30	16.69
	1	0.01	0.04	-	0.6	-	-	659	18.1	1487	12-57-31	159.87
	1	0.01	0.04	-	0.5	-	-	1107	18.12	901.83	13-54-33	642.37

Table A.6: Performance comparison of the KAF-based algorithms using normalization scaling method and sorted training examples by their sequence length in a descending (·) and ascending order [·].

Algorithm	h	λ	ν	η	$\varepsilon_{\mathbb{U}}$	μ_0	τ	m	RMSE	Score	Early - On time - Late	Training Time (sec)
ALD-KRLS	1	0.01	0.5	-	-	-	-	(75) [77]	(18.39) [18.5]	(1000) [1099.2]	(17-59-24) [16-55-29]	(1.96) [2.06]
	1	0.01	0.4	-	-	-	-	(113) [123]	(17.36) [18.16]	(707.95) [1131.4]	(19-56-25) [14-59-27]	(3.67) [4.22]
	1	0.01	0.3	-	-	-	-	(189) [194]	(17.49) [18.45]	(900.57) [1284.5]	(13-62-25) [11-56-33]	(8.59) [9.17]
	1	0.01	0.2	-	-	-	-	(341) [343]	(17.18) [18.02]	(**571.81**) [963.67]	(16-60-24) [7-58-35]	(33.42) [32.9]
	1	0.01	0.1	-	-	-	-	(793) [786]	(17.38) [18.41]	(792.51) [1461]	(16-60-24) [10-56-34]	(317.5) [294.75]
	1	0.01	0.08	-	-	-	-	(1023) [1021]	(17.47) [18.61]	(729.24) [1552.5]	(13-64-23) [11-57-32]	(631.13) [588.53]
QKRLS	1	0.01	-	-	2	-	-	(18) [20]	(23.75) [23.13]	(1483.5) [1325.4]	(33-25-42) [24-43-33]	(0.58) [0.54]
	1	0.01	-	-	1	-	-	(128) [140]	(21.61) [21.42]	(2898.2) [2450.4]	(7-49-44) [11-46-43]	(3.99) [4.45]
	1	0.01	-	-	0.8	-	-	(288) [271]	(19.74) [19.69]	(1707.6) [1670.3]	(9-49-42) [8-56-36]	(16.52) [15.44]
	1	0.01	-	-	0.6	-	-	(655) [656]	(17.94) [17.94]	(1025.7) [878.1]	(10-60-30) [11-61-28]	(156.08) [146.38]
	1	0.01	-	-	0.5	-	-	(1131) [1159]	(17.33) [17.26]	(787.32) [747.23]	(10-62-28) [11-63-26]	(663.86) [683.77]
QKLMS	1	-	-	0.01	2	-	-	(18) [20]	(22.24) [26]	(755.99) [5120.3]	(41-42-17) [11-33-56]	(0.31) [0.31]
	1	-	-	0.01	1	-	-	(128) [140]	(20.66) [24.94]	(744.81) [4405.7]	(39-46-15) [8-35-57]	(0.84) [0.86]
	1	-	-	0.01	0.8	-	-	(288) [271]	(20.67) [24.84]	(737.21) [4951.8]	(37-51-12) [5-38-57]	(1.2) [1.22]
	1	-	-	0.01	0.6	-	-	(655) [656]	(20.41) [25.06]	(707.91) [5257.9]	(39-47-14) [5-39-56]	(2.02) [2]
	1	-	-	0.01	0.5	-	-	(1131) [1159]	(20.54) [25.02]	(725.23) [5479.2]	(40-48-12) [4-40-56]	(2.87) [2.95]
	1	-	-	0.01	0.45	-	-	(1564) [1560]	(20.37) [25.1]	(699.16) [5681.7]	(39-50-11) [4-40-56]	(3.54) [3.75]
SW-KRLS	1	0.1	-	-	-	-	-	100	(28.46) [33.81]	(8962.8) [12996]	(44-40-16) [40-27-33]	(2.49) [2.46]
	1	0.1	-	-	-	-	-	200	(27.32) [30.25]	(7100.4) [12572]	(21-35-44) [10-36-54]	(5.76) [5.64]
	1	0.1	-	-	-	-	-	600	(26.16) [24.74]	(1299.9) [2998.7]	(48-40-12) [4-44-52]	(90.04) [89.58]
	1	0.1	-	-	-	-	-	770	(26.81) [24.28]	(1421) [2827.5]	(51-37-12) [3-42-55]	(145.78) [144.38]
	1	0.1	-	-	-	-	-	1000	(24.86) [23.81]	(1270) [2557]	(50-36-14) [3-43-54]	(258.62) [244.66]
	1	0.1	-	-	-	-	-	1500	(25.48) [22.98]	(1437.6) [2029.4]	(52-38-10) [14-45-41]	(533.57) [524.07]
FB-KRLS	1	0.1	-	0.00001	-	-	-	100	(23.6) [22.45]	(1372.4) [2643.6]	(42-38-20) [13-35-52]	(3.2) [3.12]
	1	0.1	-	0.00001	-	-	-	200	(23.87) [23.5]	(868.4) [4935.9]	(55-30-15) [11-44-45]	(6.31) [6.55]
	1	0.1	-	0.00001	-	-	-	450	(24.75) [23.77]	(870.54) [4930.7]	(58-30-12) [10-35-55]	(53.36) [53.78]
	1	0.1	-	0.00001	-	-	-	600	(25.72) [20.91]	(1051.7) [2772.4]	(56-35-9) [11-46-43]	(92.89) [92.51]
	1	0.1	-	0.00001	-	-	-	1000	(29.36) [21.43]	(1391.3) [2510.8]	(70-24-6) [11-44-45]	(249.77) [257.69]
	1	0.1	-	0.00001	-	-	-	1500	(31.87) [20.03]	(1825.2) [1128.3]	(74-22-4) [22-42-36]	(534.13) [543.08]
ANS-QKRLS	1	0	0.4	-	0.05	0.95	0.00001	(108) [111]	(20.21) [17.85]	(2212.9) [647.52]	(7-55-38) [20-60-20]	(16.65) [16.9]
	1	0	0.2	-	0.05	0.95	0.00001	(300) [298]	(20.67) [17.66]	(1856.2) [608.82]	(6-50-44) [20-58-22]	(62.29) [60.43]
	1	0	0.1	-	0.05	0.95	0.00001	(642) [624]	(22.44) [18.55]	(2800.5) [685.86]	(5-44-51) [23-59-18]	(265.29) [243.22]
	1	0	0.08	-	0.05	0.95	0.00001	(782) [780]	(23.01) [18.36]	(3018.5) [**537.79**]	(6-39-55) [26-55-19]	(405.46) [388.7]
	1	0	0.065	-	0.05	0.95	0.00001	(945) [940]	(22.93) [19.6]	(2531.4) [776.55]	(5-42-53) [25-55-20]	(640.65) [602.82]
	1	0	0.05	-	0.05	0.95	0.00001	(1159) [1162]	(23.52) [20.24]	(2902.7) [814.92]	(5-42-53) [28-56-16]	(1051.03) [999.44]
QALD-KRLS	1	0.01	0.4	-	0.1	-	-	(113) [123]	(17.36) [18.16]	(707.99) [1131.6]	(19-56-25) [14-59-27]	(4.3) [4.98]
	1	0.01	0.2	-	0.1	-	-	(341) [343]	(17.18) [18.02]	(**571.72**) [963.59]	(16-60-24) [7-58-35]	(36.33) [34.92]
	1	0.01	0.1	-	0.1	-	-	(793) [786]	(17.39) [18.4]	(791.24) [1461.2]	(16-60-24) [10-56-34]	(320.17) [298.88]
	1	0.01	0.04	-	2	-	-	(18) [20]	(21.2) [20.54]	(1072.1) [1371.7]	(23-49-28) [19-56-25]	(0.79) [0.73]
	1	0.01	0.04	-	1	-	-	(128) [140]	(20.62) [21.84]	(2284.9) [7863.3]	(19-55-26) [11-54-35]	(4.67) [5.02]
	1	0.01	0.04	-	0.8	-	-	(288) [271]	(19.46) [20.16]	(1558.6) [2233.9]	(13-59-28) [13-56-31]	(20.06) [16.84]
	1	0.01	0.04	-	0.6	-	-	(655) [656]	(19.43) [19.1]	(1105.1) [1369]	(16-55-29) [14-58-28]	(151.69) [139.6]
	1	0.01	0.04	-	0.5	-	-	(1111) [1126]	(18.2) [18.74]	(957.95) [977.42]	(14-62-24) [14-55-31]	(618.73) [623.46]

Table A.7: Performance comparison of the KAF-based algorithms using standardization scaling method and natural handling of engine order in the training phase.

Algorithm	h	λ	ν	η	ε_{U}	μ_0	τ	m	RMSE	Score	Early - On time - Late	Training Time (sec)
ALD-KRLS	7	0.01	0.1	-	-	-	-	77	18.3	1145.2	13-61-26	2.41
	7	0.01	0.07	-	-	-	-	120	17.92	1132.4	9-64-27	4.46
	7	0.01	0.05	-	-	-	-	180	17.68	1092.9	10-65-25	9.3
	7	0.01	0.03	-	-	-	-	355	17.88	1146.6	8-63-29	52.26
	7	0.01	0.02	-	-	-	-	736	17.88	1177.3	10-62-28	306.84
	7	0.01	0.017	-	-	-	-	1052	17.79	1121	9-65-26	767.9
QKRLS	7	0.01	-	-	25	-	-	21	19.5	861.21	19-42-39	1.1
	7	0.01	-	-	13	-	-	127	17.83	719.08	12-52-36	5.95
	7	0.01	-	-	10	-	-	307	17.77	948.69	9-65-26	25
	7	0.01	-	-	8	-	-	635	17.64	917.81	10-59-31	146.35
	7	0.01	-	-	7	-	-	966	17.64	861.9	10-66-24	468.57
	7	0.01	-	-	6.8	-	-	1059	17.51	778.89	11-66-23	574.52
	7	0.01	-	-	6.5	-	-	1190	17.1	677.49	9-67-24	778.3
QKLMS	7	-	-	0.01	25	-	-	21	19.24	779.78	17-59-24	0.31
	7	-	-	0.01	13	-	-	127	18.4	758.38	25-55-20	0.79
	7	-	-	0.01	10	-	-	307	18.52	794.75	22-58-20	1.31
	7	-	-	0.01	8	-	-	635	18.3	772.9	23-57-20	1.92
	7	-	-	0.01	7	-	-	966	18.27	772.3	22-58-20	2.6
	7	-	-	0.01	6.8	-	-	1059	18.28	756.06	23-58-19	2.78
	7	-	-	0.01	6.5	-	-	1190	18.28	751.04	23-58-19	3
	7	-	-	0.01	6	-	-	1504	18.33	795.72	23-57-20	3.65
SW-KRLS	7	0.1	-	-	-	-	-	100	24.38	993.47	45-31-24	4.07
	7	0.1	-	-	-	-	-	200	23.2	1706	20-43-37	8.77
	7	0.1	-	-	-	-	-	600	21.79	818.6	39-48-13	94.8
	7	0.1	-	-	-	-	-	770	21.26	736.8	37-49-14	151
	7	0.1	-	-	-	-	-	1000	17.85	685.44	22-59-19	248.09
	7	0.1	-	-	-	-	-	1500	17.51	949.43	10-64-26	540.8
FB-KRLS	7	0.1	-	0.00001	-	-	-	100	20.11	1500.2	20-42-38	5.35
	7	0.1	-	0.00001	-	-	-	200	20.3	1133.4	28-38-34	9.81
	7	0.1	-	0.00001	-	-	-	450	22.16	756.05	43-43-14	56.37
	7	0.1	-	0.00001	-	-	-	600	24.22	846.5	48-38-14	98.3
	7	0.1	-	0.00001	-	-	-	1000	28.39	1268.3	65-28-7	257.6
	7	0.1	-	0.00001	-	-	-	1500	35.91	2713.6	86-11-3	568.84
ANS-QKRLS	7	0	0.05	-	0.005	0.99	0.00001	114	18.19	562.17	27-56-17	23.91
	7	0	0.03	-	0.005	0.99	0.00001	160	17.58	**468.58**	24-57-19	34.53
	7	0	0.004	-	0.005	0.99	0.00001	598	20.59	644.7	35-46-19	283.77
	7	0	0.003	-	0.005	0.99	0.00001	700	20.91	664.74	37-45-18	398.71
	7	0	0.0025	-	0.005	0.99	0.00001	773	20.73	673.43	39-41-20	472
	7	0	0.0015	-	0.005	0.99	0.00001	999	20.65	652.37	37-45-18	828.46
QALD-KRLS	7	0.01	0.07	-	0.1	-	-	120	17.92	1132.4	9-64-27	5.32
	7	0.01	0.03	-	0.1	-	-	355	17.88	1146.6	8-63-29	47.81
	7	0.01	0.02	-	0.1	-	-	736	17.88	1177.3	10-62-28	296.87
	7	0.01	0.017	-	0.1	-	-	1058	17.79	1121	9-65-26	764.5
	7	0.01	0.01	-	25	-	-	21	18.35	828.42	20-53-27	0.8
	7	0.01	0.01	-	13	-	-	127	19.32	1533.6	12-54-34	4.51
	7	0.01	0.01	-	8	-	-	635	19.22	1754	11-57-32	141.47
	7	0.01	0.01	-	6.8	-	-	1059	17.73	1111.3	11-62-27	559.62

Table A.8: Performance comparison of the KAF-based algorithms using standardization scaling method and sorted training examples by their sequence length in a descending (·) and ascending order [·].

Algorithm	h	λ	ν	η	ε_{U}	μ_0	τ	m	RMSE	Score	Early - On time - Late	Training Time (sec)
ALD-KRLS	7	0.01	0.1	-	-	-	-	(76) [79]	(18.37) [19.16]	(1321.6) [1395.5]	(13-63-24) [9-55-36]	(2.26) [2.21]
	7	0.01	0.07	-	-	-	-	(116) [122]	(17.97) [19.08]	(1018.9) [1212.7]	(11-61-28) [13-54-33]	(4.13) [4.12]
	7	0.01	0.05	-	-	-	-	(181) [186]	(17.28) [17.28]	(691.07) [1259.1]	(14-63-23) [8-58-34]	(8.33) [8.52]
	7	0.01	0.03	-	-	-	-	(369) [370]	(17.07) [18.3]	(677.58) [1145]	(14-66-20) [7-61-32]	(48.26) [45.13]
	7	0.01	0.02	-	-	-	-	(777) [758]	(17.19) [18.12]	(663.46) [1205]	(14-63-23) [8-62-30]	(329.57) [294.38]
	7	0.01	0.017	-	-	-	-	(1110) [1083]	(17.39) [18.15]	(710.83) [1160.5]	(15-63-22) [8-60-32]	(842.03) [742.84]
QKRLS	7	0.01	-	-	25	-	-	(22) [23]	(19.02) [18.95]	(**601.22**) [**633.32**]	(23-43-34) [25-42-33]	(0.59) [0.59]
	7	0.01	-	-	13	-	-	(128) [131]	(18.08) [18.46]	(1017.2) [950]	(11-61-28) [10-57-33]	(4.09) [4.16]
	7	0.01	-	-	10	-	-	(297) [302]	(18.95) [18.25]	(1341.1) [901.95]	(13-54-33) [12-56-32]	(19.62) [19.38]
	7	0.01	-	-	8	-	-	(621) [623]	(17.69) [18.01]	(890.87) [880.09]	(11-61-28) [12-57-31]	(136.74) [135.05]
	7	0.01	-	-	7	-	-	(956) [943]	(17.88) [17.71]	(831.6) [801.52]	(14-59-27) [11-57-32]	(425.66) [393.35]
	7	0.01	-	-	6.8	-	-	(1040) [1038]	(17.98) [18.27]	(944.04) [896.04]	(13-58-29) [9-63-28]	(548.68) [520.42]
	7	0.01	-	-	6.5	-	-	(1203) [1197]	(17.6) [17.94]	(822.87) [952.84]	(11-65-24) [11-61-28]	(806.08) [722.42]
QKLMS	7	-	-	0.01	25	-	-	(22) [23]	(19.28) [25.36]	(**556.1**) [3098.4]	(31-52-17) [3-38-59]	(0.3) [0.3]
	7	-	-	0.01	13	-	-	(128) [131]	(20.25) [24.77]	(686.23) [3737]	(37-48-15) [6-34-60]	(0.81) [0.8]
	7	-	-	0.01	10	-	-	(297) [302]	(20.16) [24.57]	(644.3) [3705]	(36-49-15) [6-33-61]	(1.25) [1.27]
	7	-	-	0.01	8	-	-	(621) [623]	(20.31) [24.8]	(664.55) [3825.3]	(39-46-15) [6-32-62]	(1.9) [1.99]
	7	-	-	0.01	7	-	-	(956) [943]	(20.39) [24.78]	(674.41) [3751.7]	(38-48-14) [6-31-63]	(2.5) [2.58]
	7	-	-	0.01	6.8	-	-	(1040) [1038]	(20.3) [24.73]	(671.01) [3842.6]	(37-48-15) [6-31-63]	(2.63) [2.65]
	7	-	-	0.01	6.5	-	-	(1203) [1197]	(20.31) [24.71]	(679.96) [3948.1]	(35-50-15) [6-31-63]	(3.1) [3.02]
	7	-	-	0.01	6	-	-	(1521) [1485]	(20.21) [24.65]	(663.97) [3818.2]	(38-47-15) [6-31-63]	(3.58) [3.48]
SW-KRLS	7	0.1	-	-	-	-	-	100	(23.32) [23.34]	(1039.2) [1687.4]	(32-38-30) [19-34-47]	(2.52) [2.53]
	7	0.1	-	-	-	-	-	200	(32.06) [31.31]	(13580) [8916.8]	(16-31-53) [3-26-71]	(5.77) [5.74]
	7	0.1	-	-	-	-	-	600	(22.61) [25.55]	(771.55) [3567.3]	(42-44-14) [4-38-58]	(100.61) [91.88]
	7	0.1	-	-	-	-	-	770	(23.66) [25.05]	(909.5) [3310.4]	(47-39-14) [4-39-57]	(142.85) [147.82]
	7	0.1	-	-	-	-	-	1000	(22.2) [24.6]	(874.63) [3037.1]	(46-42-12) [4-39-57]	(242.83) [253]
	7	0.1	-	-	-	-	-	1500	(23.88) [23.69]	(1164.2) [2904.6]	(53-37-10) [12-46-42]	(537.33) [535.02]
FB-KRLS	7	0.1	-	0.00001	-	-	-	100	(24.32) [24.12]	(864.3) [5074.3]	(48-29-23) [7-40-53]	(3.22) [3.24]
	7	0.1	-	0.00001	-	-	-	200	(26.2) [23.13]	(1014.5) [2767.2]	(52-35-13) [8-38-54]	(6.28) [6.46]
	7	0.1	-	0.00001	-	-	-	450	(28.9) [21.12]	(1393.6) [1520.5]	(65-28-7) [24-38-38]	(53.61) [56.3]
	7	0.1	-	0.00001	-	-	-	600	(30.82) [20.92]	(1651.5) [1194.8]	(73-22-5) [27-40-33]	(98.43) [96.17]
	7	0.1	-	0.00001	-	-	-	1000	(39.12) [22.52]	(3676) [825.2]	(88-10-2) [41-41-18]	(267.93) [254.94]
	7	0.1	-	0.00001	-	-	-	1500	(43.53) [29.27]	(5755.3) [1464.8]	(95-3-2) [74-21-5]	(532.2) [618.98]
ANS-QKRLS	7	0	0.05	-	0.005	0.99	0.00001	(107) [120]	(22.39) [18.94]	(2411.5) [645.43]	(6-41-53) [24-56-20]	(17.31) [18.44]
	7	0	0.03	-	0.005	0.99	0.00001	(166) [172]	(22.62) [18.22]	(2670.6) [**626.45**]	(5-38-57) [20-59-21]	(27.79) [28.88]
	7	0	0.004	-	0.005	0.99	0.00001	(621) [617]	(24.67) [21.6]	(2888.5) [1242.7]	(4-35-61) [29-49-22]	(267.46) [255.58]
	7	0	0.003	-	0.005	0.99	0.00001	(727) [722]	(24.71) [21.56]	(3213.6) [1451.9]	(4-37-59) [29-50-21]	(373.62) [373.44]
	7	0	0.0025	-	0.005	0.99	0.00001	(794) [800]	(24.66) [20.75]	(3096.9) [846.05]	(4-41-55) [27-52-21]	(461.2) [447.04]
	7	0	0.0015	-	0.005	0.99	0.00001	(1025) [1027]	(24.89) [22.64]	(3461.9) [1490.7]	(4-43-53) [24-51-25]	(836.54) [797.43]
QALD-KRLS	7	0.01	0.07	-	0.1	-	-	(116) [122]	(17.97) [19.08]	(1018.9) [1212.7]	(11-61-28) [13-54-33]	(5.13) [5.39]
	7	0.01	0.03	-	0.1	-	-	(369) [370]	(17.07) [18.3]	(677.58) [1145]	(14-66-20) [7-61-32]	(50.39) [46.62]
	7	0.01	0.02	-	0.1	-	-	(777) [758]	(17.19) [18.12]	(663.45) [1205]	(14-63-23) [8-62-30]	(328.79) [294.48]
	7	0.01	0.017	-	0.1	-	-	(1110) [1083]	(17.39) [18.15]	(710.83) [1160.5]	(15-63-22) [8-60-32]	(843.54) [752.04]
	7	0.01	0.01	-	25	-	-	(22) [23]	(18.23) [19.25]	(**586.5**) [728.46]	(22-52-26) [24-46-30]	(0.82) [0.8]
	7	0.01	0.01	-	13	-	-	(128) [131]	(18.01) [20.43]	(1060.4) [1350]	(12-64-24) [11-56-33]	(4.77) [4.83]
	7	0.01	0.01	-	8	-	-	(621) [623]	(17.84) [18.58]	(753.7) [1148.9]	(20-60-20) [11-55-34]	(131.78) [131.2]
	7	0.01	0.01	-	6.8	-	-	(1040) [1038]	(17.81) [19.22]	(857.55) [1506.6]	(18-59-23) [8-56-36]	(515.63) [502.6]

References

[1] T. Hofmann, B. Schölkopf, A. J. Smola, Kernel methods in machine learning, The Annals of Statistics 36 (3) (2008) 1171 – 1220.
[2] J. Shawe-Taylor, N. Cristianini, et al., Kernel methods for pattern analysis, Cambridge university press, 2004.
[3] N. Aronszajn, Theory of reproducing kernels, Transactions of the American mathematical society 68 (3) (1950) 337–404.
[4] A. J. Smola, B. Schölkopf, Learning with kernels, Vol. 4, Citeseer, 1998.
[5] W. S. Noble, What is a support vector machine?, Nature biotechnology 24 (12) (2006) 1565–1567.
[6] J. Q. Shi, T. Choi, Gaussian process regression analysis for functional data, CRC press, 2011.
[7] B. Schölkopf, A. Smola, K.-R. Müller, Kernel principal component analysis, in: International conference on artificial neural networks, Springer, 1997, pp. 583–588.
[8] M. Tipping, The relevance vector machine, Advances in neural information processing systems 12 (1999).
[9] J. C. Príncipe, W. Liu, S. Haykin, Kernel adaptive filtering: a comprehensive introduction, John Wiley & Sons, 2011.
[10] S. Garcia-Vega, X.-J. Zeng, J. Keane, Stock returns prediction using kernel adaptive filtering within a stock market interdependence approach, Expert Systems with Applications 160 (2020) 113668.
[11] L. Shi, J. Tan, J. Wang, Q. Li, L. Lu, B. Chen, Robust kernel adaptive filtering for nonlinear time series prediction, Signal Processing 210 (2023) 109090.
[12] H. Zhou, J. Huang, F. Lu, Reduced kernel recursive least squares algorithm for aero-engine degradation prediction, Mechanical Systems and Signal Processing 95 (2017) 446–467.
[13] H. Zhou, J. Huang, F. Lu, J. Thiyagalingam, T. Kirubarajan, Echo state kernel recursive least squares algorithm for machine condition prediction, Mechanical Systems and Signal Processing 111 (2018) 68–86.
[14] W. Ma, J. Duan, W. Man, H. Zhao, B. Chen, Robust kernel adaptive filters based on mean p-power error for noisy chaotic time series prediction, Engineering Applications of Artificial Intelligence 58 (2017) 101–110.
[15] A. S. Eltrass, Novel cascade filter design of improved sparse low-rank matrix estimation and kernel adaptive filtering for ecg denoising and artifacts cancellation, Biomedical Signal Processing and Control 77 (2022) 103750.
[16] S. An, W. Liu, S. Venkatesh, Fast cross-validation algorithms for least squares support vector machine and kernel ridge regression, Pattern Recognition 40 (8) (2007) 2154–2162.
[17] B. Chen, J. Liang, N. Zheng, J. C. Príncipe, Kernel least mean square with adaptive kernel size, Neurocomputing 191 (2016) 95–106.
[18] M. Yukawa, Multikernel adaptive filtering, IEEE Transactions on Signal Processing 60 (9) (2012) 4672–4682.
[19] J. Zhao, H. Zhang, J. A. Zhang, Gaussian kernel adaptive filters with adaptive kernel bandwidth, Signal Processing 166 (2020) 107270.
[20] K. Slavakis, S. Theodoridis, I. Yamada, Online kernel-based classification using adaptive projection algorithms, IEEE Transactions on Signal Processing 56 (7) (2008) 2781–2796.
[21] J. Platt, A resource-allocating network for function interpolation, Neural computation 3 (2) (1991) 213–225.
[22] W. Liu, I. Park, J. C. Principe, An information theoretic approach of designing sparse kernel adaptive filters, IEEE transactions on neural networks 20 (12) (2009) 1950–1961.
[23] C. Richard, J. C. M. Bermudez, P. Honeine, Online prediction of time series data with kernels, IEEE Transactions on Signal Processing 57 (3) (2008) 1058–1067.
[24] L. Csató, M. Opper, Sparse on-line gaussian processes, Neural computation 14 (3) (2002) 641–668.
[25] Y. Engel, S. Mannor, R. Meir, The kernel recursive least-squares algorithm, IEEE Transactions on signal processing 52 (8) (2004) 2275–2285.
[26] B. Chen, S. Zhao, P. Zhu, J. C. Príncipe, Quantized kernel least mean square algorithm, IEEE Transactions on Neural Networks and Learning Systems 23 (1) (2011) 22–32.
[27] B. Chen, S. Zhao, P. Zhu, J. C. Principe, Quantized kernel recursive least squares algorithm, IEEE transactions on neural networks and learning systems 24 (9) (2013) 1484–1491.
[28] S. Van Vaerenbergh, J. Via, I. Santamaría, A sliding-window kernel rls algorithm and its application to nonlinear channel identification, in: 2006 IEEE International Conference on Acoustics Speech and Signal Processing Proceedings, Vol. 5, IEEE, 2006, pp. V–V.
[29] S. Van Vaerenbergh, I. Santamaría, W. Liu, J. C. Príncipe, Fixed-budget kernel recursive least-squares, in: 2010 IEEE International Conference on Acoustics, Speech and Signal Processing, IEEE, 2010, pp. 1882–1885.
[30] J. Guo, H. Chen, S. Chen, Improved kernel recursive least squares algorithm based online prediction for nonstationary time series, IEEE Signal Processing Letters 27 (2020) 1365–1369.
[31] M. Han, S. Zhang, M. Xu, T. Qiu, N. Wang, Multivariate chaotic time series online prediction based on improved kernel recursive least squares algorithm, IEEE transactions on cybernetics 49 (4) (2018) 1160–1172.
[32] K. Li, J. C. Principe, Transfer learning in adaptive filters: The nearest instance centroid-estimation kernel least-mean-square algorithm, IEEE Transactions on Signal Processing 65 (24) (2017) 6520–6535.
[33] S. Van Vaerenbergh, M. Lázaro-Gredilla, I. Santamaría, Kernel recursive least-squares tracker for time-varying regression, IEEE transactions on neural networks and learning systems 23 (8) (2012) 1313–1326.
[34] M. Shen, K. Xiong, S. Wang, Multikernel adaptive filtering based on random features approximation, Signal Processing 176 (2020) 107712.
[35] T. Zhang, S. Wang, X. Huang, L. Jia, Kernel recursive least squares algorithm based on the nyström method with k-means sampling, IEEE Signal Processing Letters 27 (2020) 361–365.
[36] X. Yang, Y. Mu, K. Cao, M. Lv, B. Peng, Y. Zhang, G. Wang, Robust kernel recursive adaptive filtering algorithms based on m-estimate, Signal Processing 207 (2023) 108952.
[37] J. Z. Sikorska, M. Hodkiewicz, L. Ma, Prognostic modelling options for remaining useful life estimation by industry, Mechanical systems and signal processing 25 (5) (2011) 1803–1836.
[38] X.-S. Si, W. Wang, C.-H. Hu, D.-H. Zhou, Remaining useful life estimation–a review on the statistical data driven approaches, European journal of operational research 213 (1) (2011) 1–14.
[39] J. Chen, R. Huang, Z. Chen, W. Mao, W. Li, Transfer learning algorithms for bearing remaining useful life prediction: A comprehensive review from an industrial application perspective, Mechanical Systems and Signal Processing 193 (2023) 110239.

[40] Y. Ding, M. Jia, Q. Miao, P. Huang, Remaining useful life estimation using deep metric transfer learning for kernel regression, Reliability Engineering & System Safety 212 (2021) 107583.
[41] X. Li, L. Zhang, Z. Wang, P. Dong, Remaining useful life prediction for lithium-ion batteries based on a hybrid model combining the long short-term memory and elman neural networks, Journal of Energy Storage 21 (2019) 510–518.
[42] Y. Zhang, Q. Tang, Y. Zhang, J. Wang, U. Stimming, A. A. Lee, Identifying degradation patterns of lithium ion batteries from impedance spectroscopy using machine learning, Nature communications 11 (1) (2020) 1706.
[43] A. El Mejdoubi, H. Chaoui, J. Sabor, H. Gualous, Remaining useful life prognosis of supercapacitors under temperature and voltage aging conditions, IEEE Transactions on Industrial Electronics 65 (5) (2017) 4357–4367.
[44] S. Zhao, F. Blaabjerg, H. Wang, An overview of artificial intelligence applications for power electronics, IEEE Transactions on Power Electronics 36 (4) (2020) 4633–4658.
[45] L. Viale, A. P. Daga, A. Fasana, L. Garibaldi, Least squares smoothed k-nearest neighbors online prediction of the remaining useful life of a nasa turbofan, Mechanical Systems and Signal Processing 190 (2023) 110154.
[46] A. Saxena, K. Goebel, D. Simon, N. Eklund, Damage propagation modeling for aircraft engine run-to-failure simulation, in: 2008 international conference on prognostics and health management, IEEE, 2008, pp. 1–9.
[47] B. Widrow, M. E. Hoff, Adaptive switching circuits, Tech. rep., Stanford Univ Ca Stanford Electronics Labs (1960).
[48] B. Widrow, Thinking about thinking: the discovery of the lms algorithm, IEEE Signal Processing Magazine 22 (1) (2005) 100–106.
[49] R. L. Plackett, Some theorems in least squares, Biometrika 37 (1/2) (1950) 149–157.
[50] A. H. Sayed, T. Kailath, A state-space approach to adaptive rls filtering, IEEE signal processing magazine 11 (3) (1994) 18–60.
[51] J. H. Manton, P.-O. Amblard, et al., A primer on reproducing kernel hilbert spaces, Foundations and Trends® in Signal Processing 8 (1–2) (2015) 1–126.
[52] W. Liu, P. P. Pokharel, J. C. Principe, The kernel least-mean-square algorithm, IEEE Transactions on Signal Processing 56 (2) (2008) 543–554.
[53] W. Liu, I. Park, Y. Wang, J. C. Principe, Extended kernel recursive least squares algorithm, IEEE Transactions on Signal Processing 57 (10) (2009) 3801–3814.
[54] B. Schölkopf, A. J. Smola, F. Bach, et al., Learning with kernels: support vector machines, regularization, optimization, and beyond, MIT press, 2002.
[55] J. Yuan, L. Bo, K. Wang, T. Yu, Adaptive spherical gaussian kernel in sparse bayesian learning framework for nonlinear regression, Expert Systems with Applications 36 (2) (2009) 3982–3989.
[56] X. Guo, S. Ou, M. Jiang, Y. Gao, J. Xu, Z. Cai, A new sparse kernel rls algorithm for identification of nonlinear systems, IEEE Access 9 (2021) 163165–163177.
[57] G. Sateesh Babu, P. Zhao, X.-L. Li, Deep convolutional neural network based regression approach for estimation of remaining useful life, in: Database Systems for Advanced Applications: 21st International Conference, DASFAA 2016, Dallas, TX, USA, April 16-19, 2016, Proceedings, Part I 21, Springer, 2016, pp. 214–228.
[58] X. Li, Q. Ding, J.-Q. Sun, Remaining useful life estimation in prognostics using deep convolution neural networks, Reliability Engineering & System Safety 172 (2018) 1–11.
[59] Y. Song, S. Gao, Y. Li, L. Jia, Q. Li, F. Pang, Distributed attention-based temporal convolutional network for remaining useful life prediction, IEEE Internet of Things Journal 8 (12) (2020) 9594–9602.
[60] A. Al-Dulaimi, S. Zabihi, A. Asif, A. Mohammadi, A multimodal and hybrid deep neural network model for remaining useful life estimation, Computers in industry 108 (2019) 186–196.
[61] E. Ramasso, M. Rombaut, N. Zerhouni, Joint prediction of observations and states in time-series: a partially supervised prognostics approach based on belief functions and knn, networks 4 (2013) 5.
[62] E. Ramasso, R. Gouriveau, Remaining useful life estimation by classification of predictions based on a neuro-fuzzy system and theory of belief functions, IEEE Transactions on Reliability 63 (2) (2014) 555–566.
[63] K. Javed, R. Gouriveau, N. Zerhouni, A new multivariate approach for prognostics based on extreme learning machine and fuzzy clustering, IEEE transactions on cybernetics 45 (12) (2015) 2626–2639.
[64] A. Khosravi, S. Nahavandi, D. Creighton, A. F. Atiya, Lower upper bound estimation method for construction of neural network-based prediction intervals, IEEE transactions on neural networks 22 (3) (2010) 337–346.
[65] I. de Pater, M. Mitici, Novel metrics to evaluate probabilistic remaining useful life prognostics with applications to turbofan engines, in: PHM Society European Conference, vol. 7, no. 1, 2022, pp. 96–109.
[66] H. Liu, Z. Liu, W. Jia, X. Lin, Remaining useful life prediction using a novel feature-attention-based end-to-end approach, IEEE Transactions on Industrial Informatics 17 (2) (2020) 1197–1207.
[67] A. L. Ellefsen, E. Bjørlykhaug, V. Æsøy, S. Ushakov, H. Zhang, Remaining useful life predictions for turbofan engine degradation using semi-supervised deep architecture, Reliability Engineering & System Safety 183 (2019) 240–251.
[68] C. Zhang, P. Lim, A. K. Qin, K. C. Tan, Multiobjective deep belief networks ensemble for remaining useful life estimation in prognostics, IEEE transactions on neural networks and learning systems 28 (10) (2016) 2306–2318.
[69] Y. Peng, H. Wang, J. Wang, D. Liu, X. Peng, A modified echo state network based remaining useful life estimation approach, in: 2012 IEEE Conference on Prognostics and Health Management, IEEE, 2012, pp. 1–7.
[70] K. Aggarwal, O. Atan, A. K. Farahat, C. Zhang, K. Ristovski, C. Gupta, Two birds with one network: Unifying failure event prediction and time-to-failure modeling, in: 2018 IEEE International Conference on Big Data (Big Data), IEEE, 2018, pp. 1308–1317.
[71] G. D. Karatzinis, N. A. Apostolikas, Y. S. Boutalis, G. A. Papakostas, Fuzzy cognitive networks in diverse applications using hybrid representative structures, International Journal of Fuzzy Systems (2023) 1–21.
[72] J.-Y. Wu, M. Wu, Z. Chen, X. Li, R. Yan, A joint classification-regression method for multi-stage remaining useful life prediction, Journal of Manufacturing Systems 58 (2021) 109–119.
[73] X. Zhang, Y. Dong, L. Wen, F. Lu, W. Li, Remaining useful life estimation based on a new convolutional and recurrent neural network, in: 2019 ieee 15th international conference on automation science and engineering (case), IEEE, 2019, pp. 317–322.
[74] Y. Liao, L. Zhang, C. Liu, Uncertainty prediction of remaining useful life using long short-term memory network based on bootstrap method,

in: 2018 ieee international conference on prognostics and health management (icphm), IEEE, 2018, pp. 1–8.
[75] C.-S. Hsu, J.-R. Jiang, Remaining useful life estimation using long short-term memory deep learning, in: 2018 ieee international conference on applied system invention (icasi), IEEE, 2018, pp. 58–61.
[76] S. Zheng, K. Ristovski, A. Farahat, C. Gupta, Long short-term memory network for remaining useful life estimation, in: 2017 IEEE international conference on prognostics and health management (ICPHM), IEEE, 2017, pp. 88–95.
[77] L. Fan, Y. Chai, X. Chen, Trend attention fully convolutional network for remaining useful life estimation, Reliability Engineering & System Safety 225 (2022) 108590.
[78] Z. Chen, M. Wu, R. Zhao, F. Guretno, R. Yan, X. Li, Machine remaining useful life prediction via an attention-based deep learning approach, IEEE Transactions on Industrial Electronics 68 (3) (2020) 2521–2531.
[79] J. Wang, G. Wen, S. Yang, Y. Liu, Remaining useful life estimation in prognostics using deep bidirectional lstm neural network, in: 2018 Prognostics and System Health Management Conference (PHM-Chongqing), IEEE, 2018, pp. 1037–1042.
[80] Z. Wu, S. Yu, X. Zhu, Y. Ji, M. Pecht, A weighted deep domain adaptation method for industrial fault prognostics according to prior distribution of complex working conditions, Ieee Access 7 (2019) 139802–139814.
[81] X. Chen, H. Wang, S. Lu, J. Xu, R. Yan, Remaining useful life prediction of turbofan engine using global health degradation representation in federated learning, Reliability Engineering & System Safety (2023) 109511.
[82] A. Ruiz-Tagle Palazuelos, E. L. Droguett, R. Pascual, A novel deep capsule neural network for remaining useful life estimation, Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability 234 (1) (2020) 151–167.
[83] J. Xia, Y. Feng, C. Lu, C. Fei, X. Xue, Lstm-based multi-layer self-attention method for remaining useful life estimation of mechanical systems, Engineering Failure Analysis 125 (2021) 105385.
[84] X. Zhang, Y. Guo, H. Shangguan, R. Li, X. Wu, A. Wang, Predicting remaining useful life of a machine based on embedded attention parallel networks, Mechanical Systems and Signal Processing 192 (2023) 110221.
[85] T. Li, Z. Zhao, C. Sun, R. Yan, X. Chen, Hierarchical attention graph convolutional network to fuse multi-sensor signals for remaining useful life prediction, Reliability Engineering & System Safety 215 (2021) 107878.
[86] J. Zhang, Y. Jiang, S. Wu, X. Li, H. Luo, S. Yin, Prediction of remaining useful life based on bidirectional gated recurrent unit with temporal self-attention mechanism, Reliability Engineering & System Safety 221 (2022) 108297.
[87] H. Mohamadipanah, M. Heydari, G. Chowdhary, Deep kernel recursive least-squares algorithm, Nonlinear Dynamics 104 (2021) 2515–2530.