Domain Adaptive Remaining Useful Life Prediction With Transformer

Xinyao Li, Jingjing Li, Lin Zuo, Lei Zhu, *Senior Member, IEEE*, and Heng Tao Shen, *Fellow, IEEE*

***Abstract*— Prognostic health management (PHM) has become a crucial part in building highly automated systems, whose primary task is to precisely predict the remaining useful life (RUL) of the system. Recently, deep models including convolutional neural network (CNN) and long short-term memory (LSTM) have been widely used to predict RUL. However, these models generally require sufficient labeled training data to guarantee fair performance, whereas a limited amount of labeled data overwhelmed by the abundance of unlabeled ones is the normality in industry, and the cost of full data annotation can be unaffordable. To attack this challenge, domain adaptation seeks to transfer the knowledge from a well-labeled source domain to another unlabeled target domain by mitigating their domain gap. In this article, we leverage domain adaptation for RUL prediction and propose a novel method by aligning distributions at both the feature level and the semantic level. The proposed method facilitates a large improvement of model performance and faster convergence. Besides, we propose to use the transformer as the backbone, which can capture long-term dependency more efficiently than the widely used recurrent neural network (RNN), and is thus critical for boosting the robustness of the model. We test our model on the commercial modular aero-propulsion system simulation (CMAPSS) dataset and its newly published variant new CMAPSS (N-CMAPSS) provided by National Aeronautics and Space Administration (NASA), achieving state-of-the-art results on both source-only RUL prediction and domain adaptive RUL prediction tasks.**

***Index Terms*— Domain adaptation, remaining useful life (RUL) prediction, transformer.**

I. Introduction

REMAINING useful life (RUL) indicates the general health state of a system, predicting for how many more working cycles the system can continue to operate normally [1]. From aerospace engines to lithium-ion batteries in electric cars, all these sophisticated equipment and systems need accurate RUL information for maintenance, better performance, and even avoiding catastrophic consequences. Typical methods to predict RUL include physical-based modeling [2] that makes prediction mainly based on the physical properties of the system and statistical-based methods [1] that rely on intensive past data and mathematical modeling. However, these methods can only extract stable patterns and have limited capacity, which is vulnerable when facing unknown and dynamic challenges.

Recently, data-driven deep learning methods form a popular trend for RUL prediction [3]. The most widely used models, recurrent neural network (RNN) [4] and its variants such as long short-term memory (LSTM) [5] and gated recurrent unit (GRU) [6], try to address the problem with their ability of extracting long-term dependencies in sequences. However, though LSTM and GRU are more efficient at dealing with long sequences than vanilla RNN, longer sequences still weaken the connection between two elements in distance. In the context of RUL prediction, this can lead to inaccurate prediction and vulnerability to local noise in data, thus weakening the model's robustness. The recently proposed transformer [7] has shown satisfactory performance in natural language processing and sequence modeling tasks. Its self-attention mechanism recognizes and extracts long-distance relationships, allowing the modeling of much longer sequences. We argue that the capability of capturing sufficient information from very long sequence lies at the core of RUL prediction, which is essentially a sequence prediction problem. Therefore, we propose to use the transformer as the backbone of our model to maximally mine out critical information from the sequential degradation process.

In addition, deep learning models are notoriously famous for their reliance on vast amounts of well-labeled training data to acquire fair performance. On the other hand, the performance of deep models could degrade severely facing testing data with a different distribution from the training data. These intrinsic properties of deep models could pose a threat to their practical application in real-world industry, where the abundance of unlabeled data and the existence of data distribution shift are common [8]. For example, a lithium-ion battery equipped on an electric car may work under different physical conditions (temperature, humidity, etc.) due to different geographical areas it is running in, causing its degradation pattern to alter. In this case, models trained on the original training data (from the source domain) are no longer valid, and retraining from scratch on the deployment data (from the target domain) is necessary. It is time-consuming, expensive, and sometimes impossible to collect labeled training data on the target domain.

To tackle the above challenges, domain adaptation [9], [10] seeks to transfer the knowledge from a well-labeled source domain to another unlabeled target domain by mitigating their domain gap. A plethora of works in the literature are

Manuscript received 16 May 2022; revised 22 July 2022; accepted 1 August 2022. Date of publication 22 August 2022; date of current version 9 September 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 62176042 and Grant 62073059, in part by the China Computer Federation (CCF)-Baidu Open Fund under Grant 2021PP15002000, and in part by the CCF-Tencent Open Fund under Grant RAGR20210107. The Associate Editor coordinating the review process was Dr. Dong Wang. *(Corresponding author: Jingjing Li.)*

Xinyao Li, Jingjing Li, Lin Zuo, and Heng Tao Shen are with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 611731, China (e-mail: jjl@uestc.edu.cn).

Lei Zhu is with the School of Information Science and Engineering, Shandong Normal University, Jinan 250061, China.

Digital Object Identifier 10.1109/TIM.2022.3200667

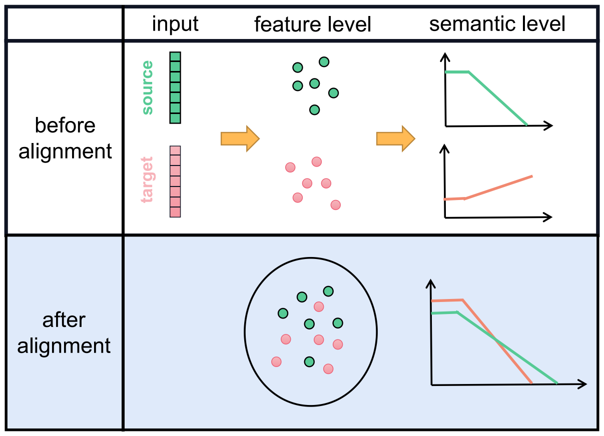

Fig. 1. Illustration of feature-level and semantic-level alignments.

feature-based methods [11], [12], which aim to learn a discriminative embedding space where the features from both the domains are well-aligned so that the knowledge learned from the former can be safely applied to the latter. The cross-entropy loss widely used in classification tasks makes predictions mainly based on the relative probabilities for each class, making it mostly immune to the change in feature scale caused by the feature-embedding process. Therefore, the model can easily map the embedded features to semantic level. However, regression tasks such as RUL prediction are sensitive to varying feature scales [13], and even slight feature fluctuation can lead to unexpected semantic outputs. To tackle the problem, we argue that semantic-level alignment is also necessary for regression tasks.

The motivation of semantic-level alignment is straightforward but effective. As illustrated in Fig. 1, perturbed features lead to unexpected semantic outputs on the target domain. These outputs can be constrained by semantic alignment, which can be regarded as a regularization term. It punishes deviating outputs like the red curve on the target domain before alignment in Fig. 1 and promotes reasonable and source-like outputs like the red curve after alignment. Besides, in practical RUL prediction tasks, most systems show similar degradation trends. They tend to perform moderately at the beginning and deteriorate faster with time. Under this premise, it is sound and reasonable to align the target outputs to the source ones.

Based on the findings above, we propose a novel domain adaption method that leverages both feature- and semantic-level alignments. The proposed method is validated on the commercial modular aero-propulsion system simulation (CMAPSS) dataset [14] and the newly released new CMAPSS (N-CMAPSS) dataset [15]. Our results show that leveraging both the transformer- and semantic-level alignments reduces root-mean-square error (RMSE) of prediction results on domain adaptive tasks by up to 63.9%. The main contributions of this article are as follows.

1) We explore domain adaptive curve for RUL prediction tasks. The proposed method can well-leverage the abundant source data for knowledge transfer between different operating conditions while demanding no labeling on target data, which saves considerable annotation costs and meets real application needs.
2) We propose a novel semantic-alignment module for cross-domain RUL prediction, allowing the model to not only extract domain-invariant features but also produce more accurate and robust semantic outputs from them, which is crucial for cross-domain RUL prediction. Instead of the widely used RNN structure, the transformer is applied to extract deep features from raw inputs for its strong capability of processing long sequences.
3) We verify the efficacy and cost of the proposed method by extensive experiments, achieving new state-of-the-art results on CMAPSS and N-CMAPSS with maximal 63.9% boost of accuracy. Besides, the confidence interval (CI) of the prediction results is quantified and illustrated. We further show that our method can be extended for online prediction.

II. Related Work

A. RUL Prediction

The RUL prediction methods can be categorized into model-based [16] and data-driven methods [17], [18], [19], [20]. The model-based methods aim to build the mathematical models according to the physical properties of the system to describe how it would work in the future [2]. However, these methods are expensive and have limited application scenarios, and thus are no longer the mainstream in RUL prediction nowadays [3]. In contrast, the data-driven methods are gaining more and more attention due to their outstanding performance. Babu *et al.* [18] applied deep CNN to RUL estimation and revealed that CNN can also handle time series forecasting. Peng *et al.* [21] proposed a Bayesian neural network for health prognostic and further enhanced it with uncertainty quantification. Recurrent models such as LSTM are able to process sequences, making them especially suitable for RUL prediction tasks. Zheng *et al.* [22] adopted LSTM for RUL prediction and gained better performance than CNN. Song *et al.* [23] proposed to apply scaled dot product attention to better extract features from sequences and combined it with bidirectional LSTM and CNN for further extraction of desired features, which reported the current state-of-the-art results. Zhang *et al.* [24] chose to use bidirectional GRU for health status assessment and RUL prediction, gaining satisfactory performance. Note that RNN variants such as GRU and LSTM can only alleviate the gradient vanishing problem in vanilla RNN [4] while processing long sequences but not eliminate it. Vaswani *et al.* [7] proposed a novel encoder–decoder model transformer, which has shown great potential in natural language processing and long sequence modeling, but is seldom used for RUL prediction tasks.

B. Domain Adaptation

Domain adaptation [9], [12] aims to transfer knowledge from a well-labeled source domain to another unlabeled target domain. The deep domain adaptation methods can generally be categorized into three types: reconstruction-based [25], [26],

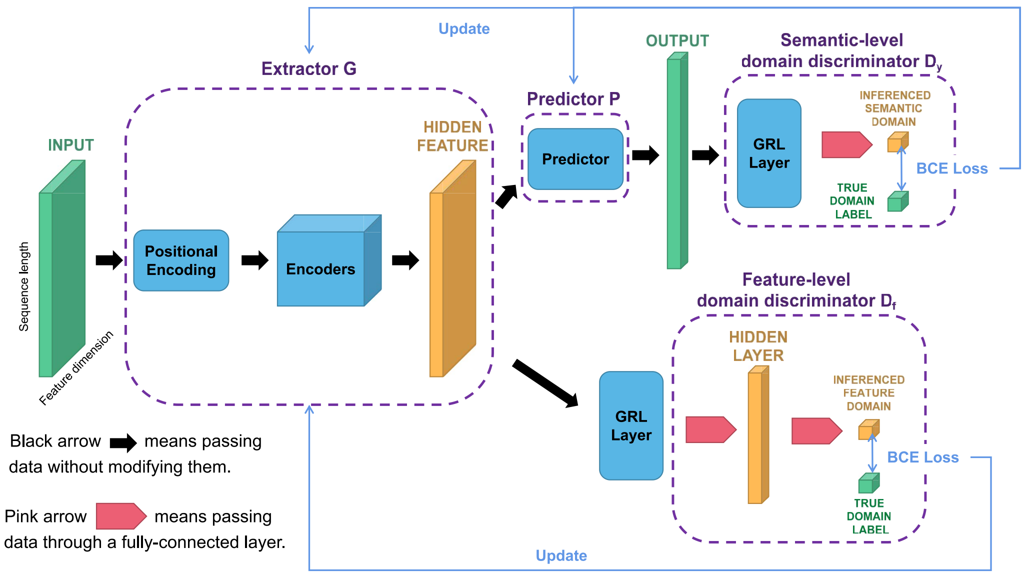
Update
Extractor G
INPUT
Sequence length
Feature dimension
Positional Encoding
Encoders
HIDDEN FEATURE
Predictor P
Predictor
OUTPUT
Semantic-level domain discriminator D_y
GRL Layer
INFERENCED SEMANTIC DOMAIN
BCE Loss
TRUE DOMAIN LABEL
Feature-level domain discriminator D_f
GRL Layer
HIDDEN LAYER
INFERENCED FEATURE DOMAIN
BCE Loss
TRUE DOMAIN LABEL
Black arrow means passing data without modifying them.
Pink arrow means passing data through a fully-connected layer.
Update

Fig. 2. Overview of our model. Input data of shape (sequence length, feature dimension) is first processed by the PE module and then passed through encoders, generating extracted hidden features, which are sent to the predictor and D_f simultaneously. Predictor P generates the corresponding RUL predictions, which serve as the output of our model. Given the output and hidden features, D_y and D_f aim to distinguish from which domain current input data are drawn. Note that GRL layers are applied before the discriminators for adversarial training.

discrepancy-based [27], [28], [29], and adversarial-based [29], [30], [31]. The reconstruction methods try to reconstruct raw data for more discriminative and common features. The discrepancy-based methods propose statistical metrics to quantify distribution discrepancy explicitly and train the model to minimize the metrics. Maximum mean discrepancy (MMD) proposed in [32] has become the most widely used distance metrics. Recently, Li *et al.* [29] proposed a novel distance metric called maximum density divergence (MDD), which minimizes the interdomain divergence and maximizes the intradomain density.

The adversarial-based methods train a domain discriminator and a feature extractor in an adversarial way to implicitly narrow domain gap. Ganin *et al.* [30] proposed a framework domain adversarial neural network (DANN) for adversarial domain adaptation, in which a gradient reversal layer (GRL) is introduced to promote adversarial training. However, these methods were originally developed for image classification, and there is yet little progress for regression tasks. One recent work carried out by Li *et al.* [33] aims to reduce the computational cost by adjusting the number of network layers according to adaptation difficulty.

For RUL prediction, researchers are paying more attention to combining popular prediction methods with domain adaptation [34], [35], [36]. Da Costa *et al.* [34] combined LSTM with the DANN framework, gaining positive knowledge transfer results. Huang *et al.* [35] propose a deep convolutional neural network (CNN)-multiple layer perceptron dual network to extract deep features and domain adaptation regularization for knowledge transfer across different machines. However, most of these methods focus solely on alignment at the feature level, neglecting adapting the semantic level, which is crucial for domain adaptive RUL prediction. Recently, there are works carried out aiming to better use target domain information. Desiring for more adaptive modeling on the target domain, we propose to further conduct alignment at the semantic level to implicitly align feature representations in both the domains, achieving new state-of-the-art results.

III. Method

A. Problem Definition

Here, we present the formal definition of the problem in this article. First, we define the RUL prediction problem. Let $X = \{x_1, x_2, \ldots, x_n\}$ where $x_i \in \mathbf{R}^{\mathbf{d}}$ represents the $d-$dimensional input features (generally are various kinds of sensor readings depending on specific application scenarios) at time step i, and $Y = \{y_1, y_2, \ldots, y_n\}$ where y_i is a scalar representing the RUL at time step i. The task is to find a mapping f so that $f(X) = Y$. For cross-domain RUL prediction, we define the source domain $D_S = \{x_S, y_S\}$ where x_S is the input data belonging to source distribution $P(X_S)$ and y_S is the corresponding label. Similarly, we have target domain $D_T = \{x_T\}$ where only unlabeled training data x_T drawn from target distribution $P(X_T)$ are available. Specifically, for domain adaptive RUL prediction tasks, we tackle the challenge from the perspective of time series modeling, so input data in the source and target domains are feature sequences with length q, i.e., $x_S \in \mathbf{R}^{q\times d}$, $x_T \in \mathbf{R}^{q\times d}$. Labels in the source domain are the corresponding RUL sequences with length q. Assuming $P(X_S) \neq P(X_T)$, the goal is to transfer knowledge from domain D_S to D_T by finding a mapping f using source data and unlabeled target data. f should work well on the target domain, i.e., $f(x_T) \approx y_T$.

B. Transformer

The transformer [7] has an encoder and a decoder. The encoder finds in-sequence relationships from input data by the self-attention mechanism and encodes the input to form hidden features. The decoder works reversely by interpreting the hidden features into desired prediction results.

1) Positional Encoding: The RNN structures pass information from one unit to another using hidden states h and inputs x, which can be formulated as $h_t, y_t = f(h_{t-1}, x_t)$, where the subscripts are time steps. From the formula, we can tell that the current output y_t is somehow related to information from previous time steps, implicitly promoting the model to extract time-related features. To further alleviate the "information fade" issue caused by long hidden states, the transformer completely deprecates the hidden state structure. It manually adds positional encoding (PE) information to input sequences so that the model can explicitly learn how relative positions in sequences affect the result. In this article, we use the same PE method as [7], which can be formulated as

$$\mathrm{PE}(\mathrm{pos}, 2i) = \sin(\mathrm{pos}/10000^{2i/d_{\mathrm{model}}}) \quad (1)$$

$$\mathrm{PE}(\mathrm{pos}, 2i+1) = \cos(\mathrm{pos}/10000^{2i/d_{\mathrm{model}}}) \quad (2)$$

where pos is the relative position of the current element in the sequence. For example, pos of element d in sequence $abcde$ is 3. i represents the ith dimension in all d dimensions. PE helps the model easily learn relative-position-based information regardless of how long the sequence is.

2) Multihead Self-Attention: Lying at the core of the transformer is the self-attention mechanism. Given keys K with the corresponding values V. For each query Q, the attention mechanism aims at computing how much attention (weight) it should pay to each value V_i, emphasizing on the most related values. In [7] and this article, relationships of pairs (Q_i, K_i) are simply computed by doing dot products between them. Formally, we have formula

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V. \quad (3)$$

The softmax function projects the weights to range (0, 1), and the division by $\sqrt{d}$ is to avoid extremely small gradients [7]. Self-attention is to compute attention within each sequence, i.e., $Q = K = V =$ processed sequence, and is computed at both the input and output ends. The transformer applies multihead attention instead of the traditional attention. Specifically

$$\text{Multihead Atten.}(Q, K, V) = \mathrm{Cat}(h_1, h_2, \ldots, h_n)W^O \quad (4)$$

$$h_i = \mathrm{Attention}\left(QW_i^Q, KW_i^K, VW_i^V\right) \quad (5)$$

where W_i^Q, W_i^K, W_i^V, and W^O are all learnable projection matrices. As shown in (4), the final output is the concatenation of all the heads.

C. Model Overview

As shown in Fig. 2, our model is composed of four parts: a feature extractor, a predictor, a feature-level domain discriminator D_f, and a semantic-level domain discriminator D_y. Assume that the batch size is 1, the input of the model should have size (l, d) representing the input sequence with length l and d features. The prediction result is a sequence with length l representing the corresponding RUL sequence predicted from the input sequence. During training, the model

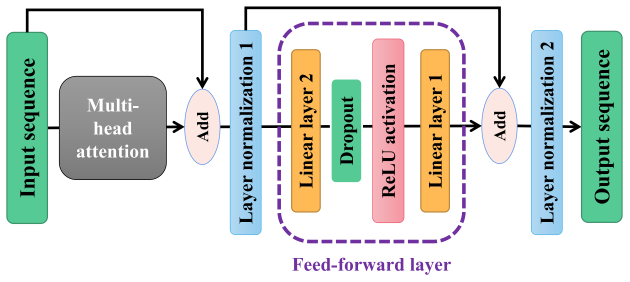

Fig. 3. Detailed structure of the encoder. The sum of input sequence and its self-attention information is normalized and serves as the input of the feed forward layer (circled by purple dotted line). The input and output of the feed forward layer are further added and normalized, forming the output sequence of the encoder.

also predicts from which domain the current data point is drawn based on the output and hidden features of the model.

1) Feature Extractor and Predictor: Feature extractor G takes sequences with length l and dimension d as the input, processes them by adding PE information pe to them, and feeds them to the encoder layers. Detailed components of an encoder can be found in Fig. 3. The hidden features extracted by the encoder are sent to predictor P for RUL prediction. P is a simple fully connected layer of size $(d, 1)$, deducing the final output from hidden features. A brief description of extractor G and predictor P can be presented as

$$\begin{aligned}\mathrm{input} &\leftarrow \mathrm{input} + \mathrm{pe}(\mathrm{input})\\ \mathrm{hidden\_features} &\leftarrow G(\mathrm{input})\\ \mathrm{output} &\leftarrow P(\mathrm{hidden\_features}).\end{aligned}$$

2) Feature-Level Domain Discriminator and Semantic-Level Domain Discriminator: The two discriminators have very similar structures and tasks. They are all composed of multilayer perceptron with one hidden layer and a *Sigmoid* activation function. They respectively output a single number $D \in (0, 1)$ representing which domain current sequence belongs to. In this article, we define source domain $D_S = 1$ and target domain $D_T = 0$. The difference is that the two discriminators make the inference based on different facts: extracted features for the feature-level discriminator and output curves for the semantic-level domain discriminator. Section III-D shows how these two discriminators work.

D. Domain Discriminators

Define input samples from the source domain as x_s with label y_s and unlabeled input samples from the target domain as x_t. The job of the feature extractor G is to extract deep feature f that is both domain-invariant and discriminative from x, i.e., $\mathrm{f} = G(x)$. To ensure that the features are discriminative, we need to ensure that the predictor is able to correctly predict the result from f. So the first job for G and P is to work together and try to correctly predict the RUL results on the source domain, since we only have access to labeled training data on the source domain. In other words, we want to minimize the prediction loss $L_y(G(x_s), y_s)$.

In this article, we select the mean square error (mse) loss as L_y. The discriminator D_f is responsible for identifying from which domain the input comes from. During training, we feed discriminator D_f with features f_s, f_t extracted from the source and target domains and the corresponding domain labels: 1 for source and 0 for target. We expect f_s and f_t to be domain-invariant by maximizing the classification loss of D_f. In other words, we consider the feature is domain-invariant if the discriminator is unable to distinguish to which domain the extracted feature belongs. We use binary cross-entropy (BCE) loss as classification loss

$$L_d^f = -\big[d_f \log \hat{d}_f + (1-d_f)\log(1-\hat{d}_f)\big] \tag{6}$$

where d_f is the true domain label for feature f and $\hat{d}_f = D_f(f)$ is the predicted domain.

However, the discriminator D_f only optimizes parameters of feature extractor G, and since we only have access to true labels of the source domain, there is no way we could directly adjust predictor P for better adaptation toward the target domain. Features from both the domains would be treated equally (both interpreted as source features) by P and some target-specific information may be neglected, causing imbalanced performance on the source and target domains. During experiments, we observe that optimizing (6) alone could cause unreasonable outputs on the target domain. For example, the predicted RUL curve may be a high horizontal line, indicating healthy system state for the entire life cycle, which is inaccurate and unreasonable. To avoid the situation, we add semantic-level domain discriminator D_y as a plug-in module. It serves as a regularization term during training by punishing the inaccurate and unreasonable outputs discussed above. For RUL prediction tasks, we know that the desired semantic outputs are similar across different domains (can be better understood referring to red curves in Fig. 4). Therefore, one simple yet effective way to realize the regularization purpose is to make output curves on both the domains undistinguishable, which happens to accord with the principle of D_f, and the difference is that D_f works on features while D_y works on outputs. Similarly, D_y is fed with output curves $\hat{y_S}$ from the source domain and $\hat{y_T}$ from the target domain. Then D_y generates one scalar $\hat{d}_y$ representing from which domain current output curve comes (1 for source and 0 for target). Finally, the classification loss of D_y is optimized by the following equation:

$$L_d^y = -\big[d_y \log \hat{d}_y + (1-d_y)\log(1-\hat{d}_y)\big] \tag{7}$$

where d_y and $\hat{d}_y$ are the true and predicted domain labels, respectively. We consider the outputs on the source and target domains is undistinguishable when classification loss L_d^y is maximized.

E. Training

Define parameters in G, P, D_f, and D_y as $\theta_G, \theta_P, \theta_{D_f}$, and θ_{D_y}, respectively, and then the overall loss can be formulated as

$$L = L_y(\theta_G, \theta_P) - \alpha L_d^f(\theta_G, \theta_{D_f}) - \beta L_d^y(\theta_G, \theta_P, \theta_{D_y}) \tag{8}$$

Algorithm 1 Model Training Algorithm

```
Input: Source domain data $X_S$, target domain data $X_T$,
       source domain label $Y_S$.
Output: Predicted label $Y_T$ for target domain.
Pretrain the model on $X_S, Y_S$.
while $epoch < max\_epoch$ do
    Randomly sample minibatches $x_s, x_t, y_s$ from
      $X_S, X_T, Y_S$.
    Extract features $\mathrm{f_s} = G(x_s)$, $\mathrm{f_t} = G(x_t)$.
    Predict results $y_s^p = P(\mathrm{f_s})$, $y_t^p = P(\mathrm{f_t})$.
    Classify domains $d_f = D_f(\mathrm{f_s}, \mathrm{f_t})$, $d_y = D_y(y_s^p, y_t^p)$.
    Compute prediction loss $L_y$ using $y_s^p, y_s$.
    Compute feature classification loss $L_d^f$ using
      $\mathrm{f_s}, \mathrm{f_t}, d_f, \hat{d}_f$.
    Compute semantic classification loss $L_d^y$ using
      $y_s^p, y_t^p, d_y, \hat{d}_y$.
    Update model parameters using (8).
end
```

where α and β are hyperparameters. Aiming to learning the most suitable parameters $\hat{\theta}_G, \hat{\theta}_P, \hat{\theta}_{D_f}$, and $\hat{\theta}_{D_y}$, we need to solve the following min-max problem:

$$\hat{\theta}_G, \hat{\theta}_P = \arg\min_{\theta_G, \theta_P} L(\theta_G, \theta_P, \hat{\theta}_{D_f}, \hat{\theta}_{D_y}) \tag{9}$$

$$\hat{\theta}_{D_f} = \arg\max_{\theta_{D_f}} L(\hat{\theta}_G, \hat{\theta}_P, \theta_{D_f}, \hat{\theta}_{D_y}) \tag{10}$$

$$\hat{\theta}_{D_y} = \arg\max_{\theta_{D_y}} L(\hat{\theta}_G, \hat{\theta}_P, \hat{\theta}_{D_f}, \theta_{D_y}). \tag{11}$$

Referring to (9)–(11), θ_G and θ_P are trained to minimize the prediction loss L_y while maximizing the feature classification loss L_d^f and output classification loss L_d^y. Inversely, θ_{D_f} and θ_{D_y} are trained to minimize L_d^f and L_d^y, respectively. The GRL proposed by Ganin *et al.* [30] provides us with an elegant way to solve the optimization problem above. Generally, gradients in the propagation path are tuned toward the same direction, which does work in most deep learning scenarios where the goal is to minimize the total loss. However, in our model we want the discriminators to be as discriminating (minimization of L_d^f and L_d^y) while features and outputs being as undistinguishable (maximization of L_d^f and L_d^y) as possible, which leads to two opposite ways. GRL solves the problem by multiplying a negative coefficient to the gradient while performing back propagation and does nothing to forward propagation. Formally, assume R to be GRL and x as the data passing through it, we can describe its behavior as

$$R(x) = x, \quad \frac{\partial R}{\partial x} = -\lambda \mathbf{I}.$$

The GRL layer is inserted between predictor P and semantic-level domain discriminator D_y, as well as feature extractor G and feature-level domain discriminator D_f, as shown in Fig. 2. During training, the feature extractor G and predictor P are pretrained for a few epochs on the source domain. The training procedure is described in Algorithm 1.

TABLE I
OVERVIEW OF THE C-MAPSS DATASET

	FD001	FD002	FD003	FD004
Engine units	100	260	100	249
Operating conditions	1	6	1	6
Fault modes	1	1	2	2
Average cycle_count	206	207	247	246

TABLE II
OVERVIEW OF THE N-CMAPSS DATASET

Name	Units	Flight classes	Failure modes	Avg cycle count
DS01	10	1,2,3	1	89
DS02	9	1,2,3	2	72
DS03	15	1,2,3	1	73
DS04	10	2,3	1	86
DS05	10	1,2,3	1	82
DS06	10	1,2,3	1	80
DS07	10	1,2,3	1	81

IV. EXPERIMENTS

A. Datasets

We select turbofan degradation datasets CMAPSS [14] and its newly released variant N-CMAPSS [15] as the benchmark for the evaluation of our model. The first dataset is widely used in the community and the second is generated from real aircraft data.

The CMAPSS dataset describes the simulated run-to-failure data of large commercial turbofan engines. According to different operating conditions and fault types, it is divided into four subsets (FD001, FD002, FD003, and FD004), each containing 24 different sensor measurements for hundreds of engine units. In Table I, we present an overview of the datasets.

N-CMAPSS is generated from real aircraft data under the same modeling strategy as CMAPSS and incorporates two new levels of fidelity, making it more challenging. Similarly, N-CMAPSS is divided into eight subsets in terms of different fault modes, namely, DS01, DS02, ..., DS08. The first seven are used for the experiment. Each subset records 38 dimensions of run-to-failure sensor data of several engine units. Flights are categorized into three classes according to their length: class 1 for flights of 1–3 h, class 2 for those of 3–5 h, and class 3 for longer flights. In Table II, we present an overview of the N-CMAPSS dataset.

B. Data Preparation

Due to different operating conditions, readings in different subsets have different scales. For smaller domain gap and smoother training process, we use min-max normalization to rescale the data to range (0, 1) by $x^i = (x^i - x^i_{\min})/(x^i_{\max} - x^i_{\min})$, where x^i is the ith sensor reading, and $x^i_{\max} and x^i_{\min}$ are the maximum and minimum values of the ith sensor reading, respectively. Missing readings are set to 1 for CMAPSS and 0 for N-CMAPSS in our experiments.

Recall that RUL represents the remaining cycle counts one can healthily operate for. That is, $\text{RUL}_c = N_{\text{total}} - c$, where N_{total} is the total cycle counts the machine can normally operate and c is the current cycle count. However, in industrial scenarios, the system degrades mildly during early life stages and wears faster in mid-to-late ones, so it is unreasonable to adopt a linear degradation pattern for every stage. Instead, we use a piecewise RUL function following previous work [37]. We set a maximum RUL limit R_C to clamp the RUL value to range $[0, R_C]$ by $\text{RUL}_c = \min\{N_{\text{total}} - c,\ R_C\}$. For CMAPSS, we set $R_C = 130$, and for N-CMAPSS, $R_C = 65$.

We apply sliding windows to generate input sequences from raw data. Due to stronger ability to process long sequences, we are able to set window size 70 for CMAPSS, which is twice or longer than most previous RNN-based works [22], [38]. Although longer sequences greatly help learn more robust features, we also note sin our experiments that increased sequence length could yield heavier reliance on PE information as well. Rather than sensor readings, the model attempts to derive the answer directly from the relative position of sequence elements, which is clearly not what we are trying to achieve. In addition to overfitting, poor generalization performance is possible as a result of only learning superficial relationships instead of real deep degradation patterns. Therefore, it is crucial to weigh the pros and cons of long sequences. Ideally, the selected sequences should possess enough deep degradation patterns while not prompting direct mapping between position and answer. In this article, after extensive experiments, we decide the sequence length to be 70 for CMAPSS and 40 for N-CMAPSS considering the reported average cycle_count in Tables I and II, respectively.

To ensure sufficient training samples at the beginning and end of an engine unit, we pad the samples if they exceed data range. Let q be the size of sliding window and x_i be the sensor readings at cycle i, we have $x_i \in \mathbf{R}^d$, where d is the sensor count. We can define sequences we sample from an engine unit j with cycle_count $= N_j$ by the following equation:

$$\text{SAMPLES}_j = \left\{\{x_k\}_{k=i-q+1}^{i}\right\}_{i=1}^{N_j+q-1} \tag{12}$$

where x_k is set to 1 if $k < 1$ or 0 if $k > N_j$. Otherwise, it remains unchanged.

Besides, for N-CMAPSS, data are recorded on a second-level granularity for each cycle, which is too small for RUL prediction. To help evoke a more distinct deterioration pattern in sequences, we average the first half and second half of each cycle, yielding data of shape (2, 38), from which deep patterns can be readily extracted with lower computational cost.

C. Evaluation Metrics

Based on each input sequence, the model generates a sequence of length q representing the predicted RUL. Consider that the sliding window step is 1, the prediction sequences may overlap with each other, resulting in multiple predictions on one position, which are later averaged for the final result. We adopt two metrics for evaluation: RMSE metric computed by RMSE $= \sqrt{\sum_{i=1}^{q}(\hat{y}_i - y_i)^2}$ and scoring algorithm by the following equation proposed by National Aeronautics and Space Administration (NASA) [39]:

$$\text{score} = \begin{cases} \sum_{i=1}^{q} e^{\frac{\hat{y}_i - y_i}{10}} - 1, & \text{if } \hat{y}_i \geq y_i \\ \sum_{i=1}^{q} e^{\frac{y_i - \hat{y}_i}{13}} - 1, & \text{if } \hat{y}_i < y_i. \end{cases} \quad (13)$$

The RMSE treats overestimating and underestimating equally, while in practical applications, the overestimated RUL could cause severer consequences. Thus, (13) assigns heavier punishment to overestimating. The metrics are computed over the final prediction result of each engine unit, and all the metrics for one subset are averaged to indicate how well the model performs.

D. Experiment Results on CMAPSS

We conduct extensive experiments on CMAPSS and N-CMAPSS using PyTorch 1.9.1 and Python 3.8.12 with a GeForce RTX 2080 Ti GPU. In this section, we present the results and chosen parameters for the source-only RUL prediction task and cross-domain RUL prediction task.

1) Source-Only RUL Prediction on CMAPSS: For this task, only feature extractor G and predictor P are necessary. We set the number of heads to eight, which is the parameter n in (4). Recall that the raw CMAPSS sensor data are 24-D, as stated in Section IV-A1. Our method requires no selection on informative sensors, and thus can be easily extended to various application scenarios even if the user has no knowledge about the system mechanism. All the sensor data (24 dimensions) are fed to the model, and therefore, we have shape (24, 512) for Linear layer 1, where 512 is the dimension of the hidden layer. Linear layer 2 is of shape (512, 24), outputting encoded 24-D data. The dropout rate is set to 0.1 to alleviate the overfitting problem. Predictor P is a linear layer of shape (24, 1) that works as a decoder, decoding the 24-D encoded data into the desired output. We divide the data into a training set containing 70% data, a test set containing 18% data, and a validation set with 12% data. For each epoch, we randomly sample a minibatch with a batch size of 128 from the training set and minimize the overall loss using a stochastic gradient descent (SGD) optimizer. The initial learning rate is set to 0.02 and drops by 50% for every 50 epochs. We train the model for 200 epochs on each subset and save the model with the best performance on the validation set. Table III shows comparison of RMSE of our method and competing methods.

In Table III, we compare our method with two shallow methods and five recent deep methods, gaining new state-of-the-art result on every domain with a maximal 31.4% boost of performance. Subset FD004 is considered most challenging with six operating conditions and two fault modes. Our method achieves the most significant improvement on FD004 by decreasing prediction RMSE from 18.1 to 12.41. Previous methods with relatively good performance were mostly constructed on variants of RNN, showing fair performance. However, due to their inherent flaw, substantial improvements are harder to obtain, whereas our transformer-based model outperforms all RNN-based methods by a wide margin, demonstrating its great potential. In addition, we can observe that the results between subsets FD001 and FD003, and FD002 and FD004 are similar, implying smaller domain shift between them. In other words, it is easier for models to transfer knowledge between these pairs, which is better illustrated in Section IV-D2.

TABLE III
EXPERIMENT RESULTS OF THE SOURCE-ONLY TASK ON CMAPSS

Method	FD001	FD002	FD003	FD004
SVR [18]	20.96	42.00	21.05	45.35
RVR [18]	23.80	31.30	22.37	34.34
CNN [18]	18.45	30.29	19.82	29.16
Deep-LSTM [22]	16.14	24.49	16.18	28.17
BiLSTM-ED [40]	14.74	22.07	17.48	23.49
DCNN [41]	12.61	22.36	12.64	23.31
Attn-Based BiLSTM-CNN [23]	12.13	16.01	11.96	18.10
Ours	**8.67**	**12.42**	**9.37**	**12.41**

2) Cross-Domain RUL Prediction on CMAPSS: In this section, we show the results on the cross-domain RUL prediction tasks. We implement D_y as a multilayer perceptron with one hidden layer of 512 neurons. Before the input is sent to the hidden layer, a ReLU activation layer is applied. The result is passed through a Sigmoid activation layer to generate an output within range (0, 1) for the computation of classification loss in (7). For feature-level domain discriminator D_f, we first add a fully connected layer to each row of length 24, reducing the 2-D matrix into a 1-D sequence of length q, and the remaining implementation is identical to that of D_y.

We conduct experiment on every cross-domain scenario, which are 12 sets of experiments in total. We train each set of experiment for 240 epochs using an SGD optimizer with a batch size of 128. The initial learning rate is 0.02 and drops by 50% every 80 epochs. The hyperparameters in (8) are presented in appendixes. Both the hyperparameters α and β are decided by grid search. Each set of hyperparameters is evaluated by cross-validation to obtain optimal choice. In Section IV-H, we will further analyze the effects of both the hyperparameters. In Tables IV and V, we compare our method with some traditional domain adaptation methods and recent RUL-oriented adaptive methods on metrics RMSE and NASA score, respectively. In addition, the results on transformer+DANN are shown in the ablation study in column "D_f Only." The results in bold font are the best among the compared methods and underlined ones are the second best. We also report the results for source-only and target-supervised tasks, which are in darker background color. A source-only model is trained only on the source domain and is applied directly to the target domain, serving as the baseline. Target-supervised, on the contrary, are the results of supervised learning on the target domains, which are the same as reported in Table III. The results of the target-supervised tasks serve as the upper bound theoretically, since domain adaptation can only narrow domain gap, not to extinguish it. In addition,

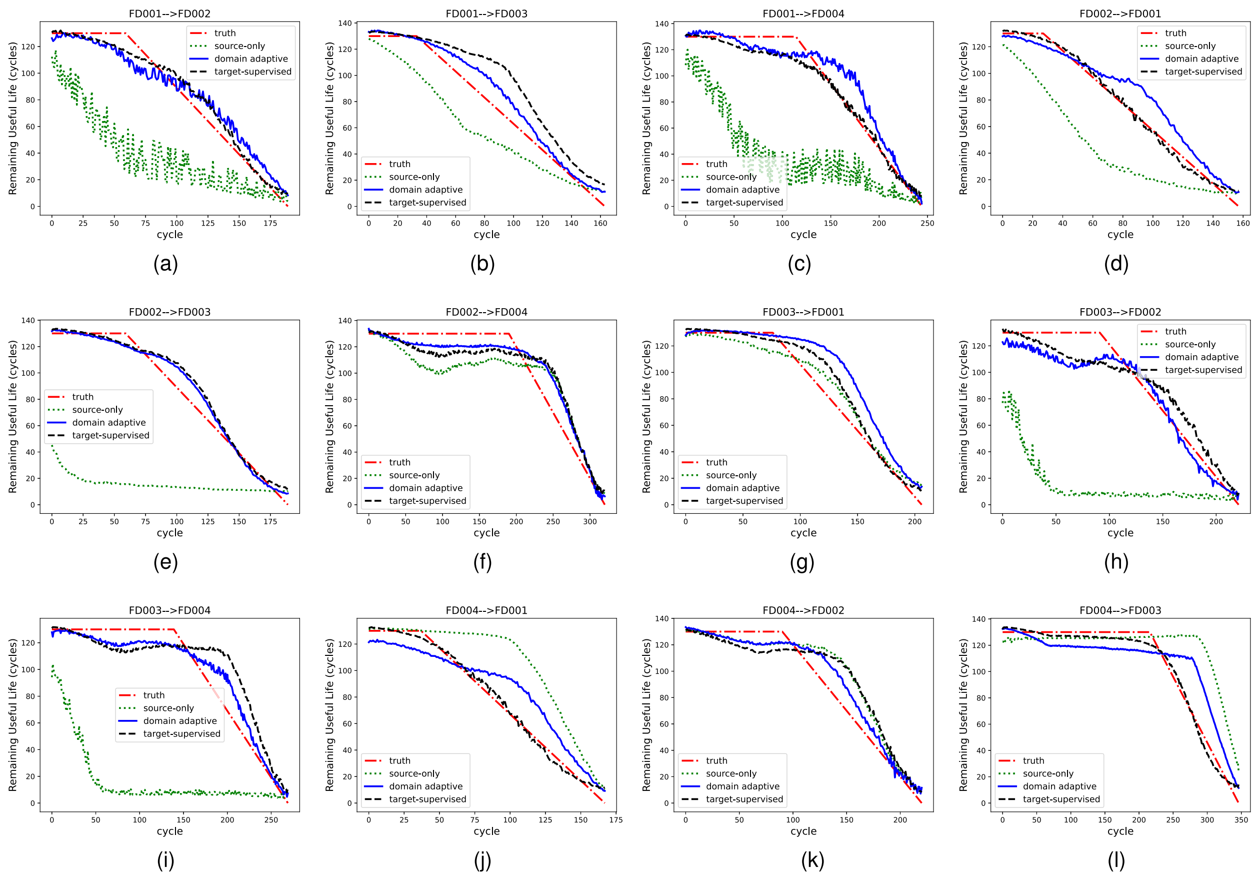

Fig. 4. Results of cross-domain RUL prediction tasks on the CMAPSS dataset. Red curves represent ground truth, green ones represent models trained on source domain only (baseline), black ones represent models trained on target domain under supervision (target-supervised), and blue ones are models after domain adaptation. (a) FD001–>FD002. (b) FD001–>FD003. (c) FD001–>FD004. (d) FD002–>FD001. (e) FD002–>FD003. (f) FD002–>FD004. (g) FD003–>FD001. (h) FD003–>FD002. (i) FD003–>FD004. (j) FD004–>FD001. (k) FD004–>FD002. (l) FD004–>FD003.

TABLE IV

RMSE RESULTS OF CROSS-DOMAIN RUL PREDICTION ON CMAPSS

Task	Improvement	Ours	CADA [42]	AdaBN [43]	LSTM+DANN [34]	ADDA [44]	Source-Only	Target-Supervised
FD001–>FD002	22.08%	**15.21**	19.52	30.54	48.6	31.26	47.44	12.42
FD001–>FD003	60.84%	**15.50**	39.58	18.40	45.9	57.09	31.90	9.37
FD001–>FD004	49.12%	**15.89**	31.23	36.85	43.8	56.66	55.81	12.41
FD002–>FD001	18.66%	**11.29**	13.88	20.66	28.1	19.73	26.25	8.67
FD002–>FD003	58.96%	**13.76**	33.53	29.24	37.5	37.22	72.95	9.37
FD002–>FD004	63.93%	**12.16**	33.71	29.64	31.8	37.64	14.38	12.41
FD003–>FD001	45.96%	**10.56**	19.54	19.86	31.7	40.41	14.51	8.67
FD003–>FD002	11.64%	**17.08**	19.33	40.39	44.6	42.53	67.89	12.42
FD003–>FD004	17.27%	**17.05**	20.61	39.97	47.9	31.88	74.80	12.41
FD004–>FD001	41.64%	**11.73**	20.10	21.71	31.5	37.81	23.65	8.67
FD004–>FD002	35.08%	**12.01**	18.5	24.01	24.9	36.67	11.97	12.42
FD004–>FD003	-2.29%	14.83	**14.49**	35.03	27.8	23.59	20.30	9.37

Results in bold font are the best among all compared methods, and underlined results are the second best.
In dark background color are source-only and target-supervised results, serving as baseline and upper bound respectively.

we show by what margin our method outperforms the recent state-of-the-art method CADA in column "Improvement."

The results on RMSE and scoring function are reported in Tables IV and V. Our method outperforms all the four competing methods significantly on RMSE except for task "FD004–>FD003" where our method is slightly behind CADA. We achieve the greatest accuracy boost on task FD002–>FD004, gaining 63.9% improvement. The source-only result on task FD002–>FD004 already exceeds all the competing methods, and aligning both feature and semantic space further improves the accuracy, reaching target-supervised results. In fact, the source-only model also gains satisfying results in other three cross-domain scenarios with relatively small domain gap, i.e., FD001–>FD003,

TABLE V
SCORE RESULTS OF CROSS-DOMAIN RUL PREDICTION ON CMAPSS

Task	Improvement	Ours	CADA [42]	AdaBN [43]	LSTM+DANN [34]	ADDA [44]	Source-Only	Target-Supervised
FD001–>FD002	47.55%	**1113**	2122	13400	93841	4865	105358	740
FD001–>FD003	66.80%	**2794**	8415	1680	27005	32472	10066	571
FD001–>FD004	88.43%	**1339**	11577	100000	57044	68859	222991	991
FD002–>FD001	-34.88%	539	**351**	1010	8411	689	12991	343
FD002–>FD003	63.19%	**1919**	5213	7000	17406	11029	650282	571
FD002–>FD004	93.17%	**1031**	15106	14100	66305	16856	1095	991
FD003–>FD001	54.24%	**664**	1451	1920	5113	32451	1320	343
FD003–>FD002	64.71%	**1855**	5257	78500	37297	459911	600476	740
FD003–>FD004	41.63%	**1879**	3219	134000	141117	82520	>1e6	991
FD004–>FD001	65.82%	**629**	1840	1580	7586	43794	8474	343
FD004–>FD002	81.01%	**847**	4460	4760	17001	23822	859	740
FD004–>FD003	-65.42%	1972	**682**	17700	5941	1117	16082	571

Results in bold font are the best among all compared methods, and underlined results are the second best.
In dark background color are source-only and target-supervised results, serving as baseline and upper bound respectively.

TABLE VI
RMSE AND SCORE RESULTS OF CROSS-DOMAIN RUL PREDICTION ON N-CMAPSS

RMSE / SCORE			Source Domain						
			DS01	DS02	DS03	DS04	DS05	DS06	DS07
Target Domain	DS01	Source-Only	4.57 / 74	9.58 / 244	31.60 / 3901	36.59 / 6463	36.98 / 6753	39.46 / 8536	38.98 / 8337
		After DA		8.83 / 206	10.75 / 291	7.51 / 155	9.93 / 244	8.83 / 206	6.97 / 147
	DS02	Source-Only	4.83 / 87	5.34 / 91	12.02 / 1058	29.14 / 3600	31.45 / 3294	33.23 / 3978	14.60 / 2030
		After DA	4.16 / 62		5.23 / 92	8.67 / 210	6.70 / 134	7.19 / 153	8.54 / 204
	DS03	Source-Only	5.53 / 101	6.88 / 149	6.73 / 145	32.35 / 3980	32.90 / 4168	34.61 / 4964	4.88 / 82
		After DA	7.13 / 173	5.18 / 93		6.42 / 141	5.44 / 109	5.05 / 94	5.57 / 107
	DS04	Source-Only	31.65 / 4114	30.45 / 3590	32.02 / 4284	9.25 / 209	34.34 / 5322	35.50 / 5938	32.02 / 4363
		After DA	8.53 / 210	9.20 / 239	9.75 / 251		8.25 / 197	8.15 / 199	9.28 / 256
	DS05	Source-Only	34.69 / 5422	37.32 / 7038	34.00 / 5143	32.30 / 4155	5.18 / 99	28.44 / 2835	34.97 / 5835
		After DA	5.58 / 105	7.34 / 166	6.10 / 111	5.52 / 109		4.62 / 80	6.15 / 127
	DS06	Source-Only	33.15 / 4336	37.34 / 6634	34.83 / 5288	27.68 / 2390	11.58 / 464	4.08 / 69	34.34 / 5155
		After DA	4.84 / 84	5.43 / 93	6.05 / 108	5.04 / 95	4.69 / 79		5.16 / 93
	DS07	Source-Only	35.55 / 5465	36.17 / 5720	21.52 / 1170	37.91 / 6923	37.79 / 6799	38.50 / 7263	6.71 / 143
		After DA	6.16 / 119	11.18 / 316	6.74 / 135	6.89 / 139	6.25 / 126	5.94 / 113	

FD003–>FD001, and FD004–>FD002, outperforming most competing methods. In tasks with large domain gap, the source-only results are undesirable due to different feature distribution. After feature and semantic alignments, the result is rather close to the target-supervised cases, suggesting the feature and semantic spaces are well-aligned. It is worth noting that since the scoring function in (13) penalizes inaccuracy exponentially, some units with relatively large error could affect the overall result significantly, resulting in great fluctuation shown in Table V.

In addition, our method shows steady performance and generates reasonable output over every cross-domain scenario. In Fig. 4, we select some representative examples from each cross-domain task for visualized comparison. As stated in Section IV-D1, relatively small domain gap exists between domains FD001 and FD003 as well as FD002 and FD004. Therefore, in these cases the source-only models tend to perform similar to the target-supervised ones [see Fig. 4(b), (f), (g), and (k)]. While in the remaining instances, the domain gap is large due to different operating conditions (1 for FD001 and FD003, and 6 for FD002 and FD004, as shown in Table I). Therefore, the source-only models show severe performance drop, resulting in large deviations in prediction curves (see Fig. 4(e), (h), (i), and so on). For these cross-domain tasks with large domain gap, the blue prediction curves after domain adaptation are sound and accurate, reaching performance of fully supervised learning, which strongly supports the advancement of our method and proves that the regularization provided by semantic alignment is effective.

E. Experiment Results on N-CMAPSS

In addition, we conduct similar experiments on the N-CMAPSS dataset to demonstrate the validity and robustness of our method. The subsets involved in this experiment (DS01, DS02, ..., DS07) are divided according to the failure modes that affect them. Failure modes include efficiency and flow degradation of fan, low pressure compressor (LPC), high pressure compressor (HPC), high pressure turbine (HPT), and low pressure turbine (LPT). Different subsets degrade for different reasons, demanding higher robustness and generalization performance of the model.

We set R_C of the piecewise function to 65, the length for each sequence to be 40, and the head number of multihead attention to be 8. We conduct 42 cross-domain RUL prediction tasks (DS01–>DS02, DS01–>DS03, etc.) To better illustrate the effectiveness of domain adaptation, we record the source-only results as baseline. It is worth mentioning that we concatenate two "fake" features with all zero values to the 38 true features, forming 40 features in total, so that each one of the eight heads can process features with the same dimension 5. Thus, the inputs of our model are sequences with shape (40, 40), representing sequences of length 40 with 40 dimensions of features.

1) Source-Only RUL Prediction on N-CMAPSS: We first conduct source-only prediction tasks for all seven subsets. We use 70% of each subset for training, 20% for testing, and 10% for validation. For all the tasks, we train the model for 180 epochs with a minibatch size of 32 using the SGD optimizer. The initial learning rate is set to 0.02 and it degrades by 50% every 60 epochs. In Table VI, we present the results of the source-only task on RMSE and score metrics in "Source-Only" rows with light-colored background. When the source domain and target domain are identical, the result of supervised learning is presented. For all seven subsets, the RMSE of supervised learning is smaller than 10, indicating high accuracy of our method. Then we directly apply the models to other domains, generating source-only results. We can observe that the models trained for DS01, DS02, or DS03 also work well on other two domains, indicating relatively small domain gap among them. On the contrary, domain shift among the other four subsets is much more significant. The RMSE of tasks involving DS04, DS05, DS06, and DS07 is greater than 30, which is unacceptable since the RUL is at most 65.

2) Cross-Domain RUL Prediction on N-CMAPSS: We perform domain adaptation on all 42 cross-domain tasks. For all the tasks, we train the model for 180 epochs with a minibatch size of 64 using the SGD optimizer. The initial learning rate is 0.015 and it drops by 50% every 90 epochs. The chosen hyperparameters in (8) are shown in appendixes. The cross-domain results are shown in Table VI with dark background color. We can see that our method significantly reduces the prediction error caused by domain gap, and in most cases the result is less than 10, approaching the accuracy of the target-supervised tasks. There are a few tasks, for example, DS01–>DS03, where prediction error slightly rises after domain adaptation. The main cause is that both the domains were originally aligned, and little domain shift exists in these tasks. Performing domain adaptation in this scenario may disturb the aligned feature distribution and cause negative optimization. As the results show, this disturbance has very limited influence on the final results and is thus acceptable.

F. Ablation Study

In this section, we present the ablation study of our method to further illustrate its validity and analyze how each part contributes to the final result. The study is conducted on the CMAPSS dataset for both the metrics, as shown in Table VII. Recall that our method consists of two discriminators D_f and D_y, responsible for feature- and semantic-level alignments, respectively. We validate the necessity of both the discriminators by disabling either one and evaluate the model for each cross-domain scenario, recorded as D_f only and D_y only in the tables. It is worth noting that D_f only is actually the incorporation of the transformer and DANN, which additionally compares our semantic alignment module and other domain adaptation module, showing the efficacy of our method.

TABLE VII
RMSE/SCORE RESULTS OF ABLATION STUDY. COLUMN "D_f ONLY" PRESENTS RESULTS ON TRANSFORMER+DANN

Task	Both	D_f Only	D_y Only
FD001–>FD002	15.21 / **1113**	15.37 / 1138	**15.14** / 1157
FD001–>FD003	**15.50** / **2794**	22.06 / 7157	17.38 / 9677
FD001–>FD004	**15.89** / **1339**	25.33 / 5735	16.14 / 1524
FD002–>FD001	**11.29** / **539**	13.05 / 788	11.87 / 614
FD002–>FD003	**13.76** / **1919**	17.29 / 9706	14.59 / 2528
FD002–>FD004	**12.16** / 1031	12.87 / 1136	12.56 / **992**
FD003–>FD001	10.56 / 664	10.61 / **581**	**10.46** / 599
FD003–>FD002	**17.08** / 1855	17.65 / **1627**	17.27 / 1958
FD003–>FD004	17.05 / **1879**	**16.99** / 2338	17.78 / 2729
FD004–>FD001	11.73 / 629	**11.68** / **618**	14.39 / 1148
FD004–>FD002	12.01 / 847	**11.57** / **790**	11.98 / 833
FD004–>FD003	**14.83** / **1972**	15.95 / 3287	15.26 / 2335

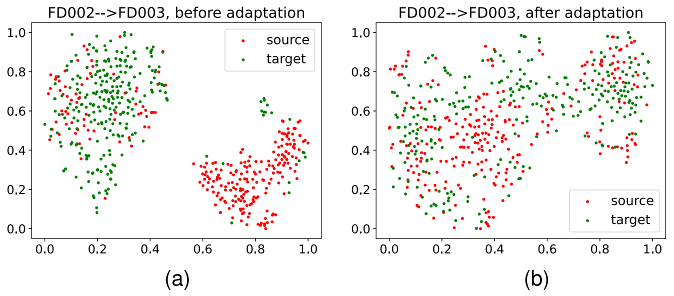

Fig. 5. Visualization of effects of feature-level alignment. Red points represent source data and green ones represent target data. (a) Without feature alignment. (b) With feature alignment.

G. Visualization

We present t-distributed stochastic neighbor embedding (t-SNE) [45] visualization on both the feature and semantic levels to intuitively demonstrate the validness of our method. As shown in Fig. 5(a), features extracted by the source model (without adaptation) form two distinct clusters for the source data and target data, implying that data distribution shift naturally exists among different domains. After feature-level alignment, however, the model is able to close the domain gap by extracting domain-invariant features. As depicted in Fig. 5(b), data points from both the domains are well-mixed and show no distinct characteristic and are thus aligned. Fig. 6 shows visualization of semantic outputs. As shown in Fig. 6(a), the source model only generates meaningful outputs on the source data (red points), while for the target data (green points) the semantic distribution is chaotic and meaningless. After semantic-level alignment, as shown in Fig. 6(b), the model generates similar and meaningful outputs for both the domains, indicating that our semantic-level alignment module is effective and sound.

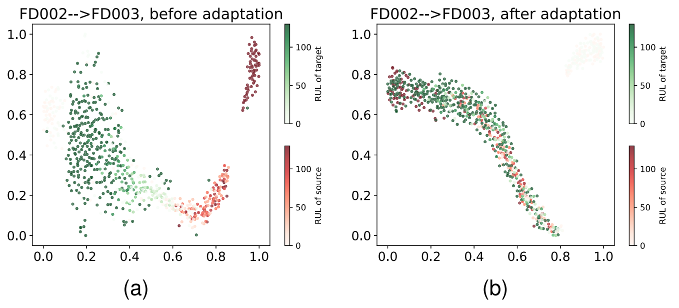

Fig. 6. Visualization of effects of semantic-level alignment. Red points represent source data and green ones represent target data. Darker color means bigger RUL. (a) Without feature alignment. (b) With feature alignment.

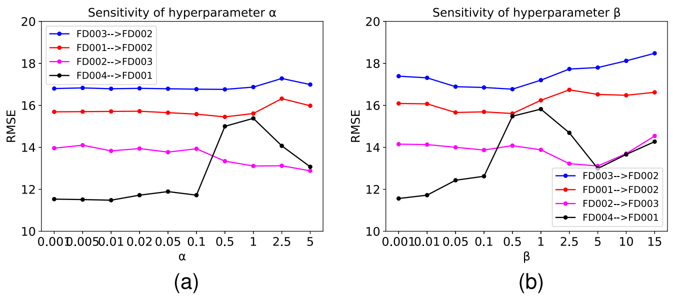

Fig. 7. Sensitivity to both the hyperparameters, indicating how prediction error changes with them. (a) Sensitivity of α. (b) Sensitivity of β.

H. Parameter Sensitivity

In this section, we analyze how the change in hyperparameters α and β in (8) affects our model. We first set either parameter to optimal state, and then test how the results change with another parameter. We choose 4 out of 12 cross-domain tasks on CMAPSS to illustrate the results.

1) Parameter β: As shown in (8), β mainly controls the model parameters optimized by the output discriminator. We test the results with β ranging from 0.001 to 15. Since D_y directly optimizes the predictor P, it has the greatest impact on the final output, and thus the results are relatively more linked to parameter β. As shown in Fig. 7(b), the results fluctuate moderately, and different tasks are best carried out with different β. Generally speaking, the fluctuation is within a small range and does not affect the overall performance dramatically.

2) Parameter α: As shown in (8), α mainly controls the alignment of extracted features, which is expected to have slighter effects on the final output than β. We test the results with α ranging from 0.001 to 5. As shown in Fig. 7(a), the results are much less affected by α compared with β, resulting in smoother fluctuations, which meets our expectations.

I. Uncertainty Quantification

In practical applications, uncertainty quantification is crucial due to safety reasons. However, there exists few works about cross-domain RUL prediction that address the problem. In this section, we use a Bayesian-based method proposed by Peng *et al.* [21] to obtain CIs of our prediction results. As proved in [46], the dropout layers in neural networks can be used to model uncertainty. Therefore, we activate the dropout layers during testing and record the results from multiple runs (set to 100 in this article) as posterior estimations of RUL. We visualize two cross-domain tasks in Fig. 8, and uncertainty quantification results for task FD004–>FD002 are listed in Table VIII. As shown in Fig. 8 and Table VIII, the model shows steady performance across domains, and the overall accuracy is acceptable.

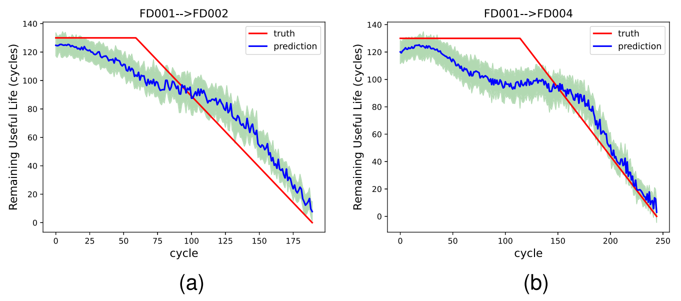

Fig. 8. Visualization for uncertainty quantification. Green shades represent CIs. (a) FD001–>FD002. (b) FD001–>FD004.

TABLE VIII
UNCERTAINTY QUANTIFICATION FOR TASK FD004–>FD002

Cycle	True RUL	Predicted RUL	95% CI
50	130	126	[122, 132]
100	120	119	[115, 123]
150	70	87	[84, 91]
180	40	50	[46, 54]
210	10	12	[8, 19]

TABLE IX
ONLINE TEST RESULTS (RMSE) FOR CROSS-DOMAIN RUL PREDICTION TASKS ON CMAPSS

Task	UDA	Online	Train Time (s)	Inference Time (s)
FD001–>FD002	15.21	23.89	1296	124
FD001–>FD003	15.50	20.31	1062	71
FD001–>FD004	15.89	37.62	1287	134
FD002–>FD001	11.29	17.36	1091	61
FD002–>FD003	13.76	17.97	1267	73
FD002–>FD004	12.16	17.74	2885	164
FD003–>FD001	10.56	16.99	1098	59
FD003–>FD002	17.08	24.60	1325	137
FD003–>FD004	17.05	25.04	1378	140
FD004–>FD001	11.73	17.21	1019	56
FD004–>FD002	12.01	15.89	2765	152
FD004–>FD003	14.83	21.46	1160	67

J. Online Test

To extend our method to multiple application scenarios, we test its performance for online tasks. Unlike the traditional unsupervised domain adaptation (UDA) tasks that are transductive, we split the target domain into train and online test sets (50% for training and 50% for testing in this article), where samples in the test sets are unknown during training. During online testing, the model keeps updating itself and learns additional knowledge from test samples. Note that for online tasks, the test samples are unknown and are normalized according to the minimum and maximum values of the training sets by min-max normalization. We further provide training and inference time for each cross-domain scenario. The results can be found in Table IX.

As shown in Table IX, the online test accuracies are generally lower than the UDA results. The reason is that our method follows classical UDA problem setting, where a trained and static model is used for testing. However, in most cases accuracy degradation is within acceptable range, demonstrating that our method can be easily extended to online prediction applications. Our future work would include improving our framework for higher accuracy on online prediction tasks and more compatibility. In Table IX, we also provide training and inference for each task. Note that the inference time includes predicting **all** the testing samples in the target domain. As shown in Table IX, the inference time for each sample is less than 1 s, and is therefore acceptable.

V. Conclusion

In this article, we propose a novel approach named semantic-level alignment for the domain adaptive RUL prediction tasks. To avoid large deviations of output curves caused by scaled features, the semantic-level alignment module serves as a regularization term that punishes unreasonable and inaccurate output curves. Feature-level alignment is also applied to extract discriminative and domain-invariant features. Besides, we adopt transformer as our backbone instead of the widely used RNN-based models, obtaining significant performance boost on the RUL prediction tasks. We verify the effectiveness of our model on CMAPSS and the newly released N-CMAPSS, achieving new state-of-the-art results. Furthermore, we also provide uncertainty quantification results of the proposed method and experimental results of online tasks. The results show the steadiness and flexibility of the proposed method.

References

[1] X.-S. Si, W. Wang, C.-H. Hu, and D.-H. Zhou, "Remaining useful life estimation—A review on the statistical data driven approaches," *Eur. J. Oper. Res.*, vol. 213, pp. 1–14, Aug. 2011.

[2] A. Cubillo, S. Perinpanayagam, and M. Esperon-Miguez, "A review of physics-based models in prognostics: Application to gears and bearings of rotating machinery," *Adv. Mech. Eng.*, vol. 8, no. 8, 2016, Art. no. 1687814016664660.

[3] Y. Wang, Y. Zhao, and S. Addepalli, "Remaining useful life prediction using deep learning approaches: A review," *Proc. Manuf.*, vol. 49, pp. 81–88, Jan. 2020.

[4] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.

[5] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[6] K. Cho *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," 2014, *arXiv:1406.1078*.

[7] A. Vaswani *et al.*, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 5998–6008.

[8] S. Vollert and A. Theissler, "Challenges of machine learning-based RUL prognosis: A review on NASA's C-MAPSS data set," in *Proc. 26th IEEE Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2021, pp. 1–8.

[9] W. M. Kouw and M. Loog, "A review of domain adaptation without target labels," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 3, pp. 766–785, Mar. 2021.

[10] W. Mei and D. Weihong, "Deep visual domain adaptation: A survey," *Neurocomputing*, vol. 312, pp. 135–153, Jul. 2018.

[11] M. Long, Y. Cao, J. Wang, and M. Jordan, "Learning transferable features with deep adaptation networks," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 97–105.

[12] J. Li, K. Lu, Z. Huang, L. Zhu, and H. T. Shen, "Transfer independently together: A generalized framework for domain adaptation," *IEEE Trans. Cybern.*, vol. 49, no. 6, pp. 2144–2155, Jun. 2019.

[13] X. Chen, S. Wang, J. Wang, and M. Long, "Representation subspace distance for domain adaptation regression," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 1749–1759.

[14] A. Saxena and K. Goebel. (2008). *Turbofan Engine Degradation Simulation Data Set*. [Online]. Available: http://ti.arc.nasa.gov/project/prognostic-data-repository

[15] M. A. Chao, C. Kulkarni, K. Goebel, and O. Fink, "Aircraft engine run-to-failure dataset under real flight conditions for prognostics and diagnostics," *Data*, vol. 6, no. 1, p. 5, Jan. 2021.

[16] M. J. Daigle and K. Goebel, "Model-based prognostics with concurrent damage progression processes," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 43, no. 3, pp. 535–546, May 2013.

[17] Y. Cheng, J. Wu, H. Zhu, S. Wing Or, and X. Shao, "Remaining useful life prognosis based on ensemble long short-term memory neural network," *IEEE Trans. Instrum. Meas.*, vol. 70, pp. 1–12, 2021.

[18] G. S. Babu, P. Zhao, and X.-L. Li, "Deep convolutional neural network based regression approach for estimation of remaining useful life," in *Proc. Int. Conf. Database Syst. Adv. Appl.* Cham, Switzerland: Springer, 2016, pp. 214–228.

[19] C. Chen, N. Lu, B. Jiang, Y. Xing, and Z. H. Zhu, "Prediction interval estimation of aeroengine remaining useful life based on bidirectional long short-term memory network," *IEEE Trans. Instrum. Meas.*, vol. 70, pp. 1–13, 2021.

[20] K. Liu, Y. Shang, Q. Ouyang, and W. D. Widanage, "A data-driven approach with uncertainty quantification for predicting future capacities and remaining useful life of lithium-ion battery," *IEEE Trans. Ind. Electron.*, vol. 68, no. 4, pp. 3170–3180, Apr. 2021.

[21] W. Peng, Z. S. Ye, and N. Chen, "Bayesian deep-learning-based health prognostics toward prognostics uncertainty," *IEEE Trans. Ind. Electron.*, vol. 67, no. 3, pp. 2283–2293, Apr. 2019.

[22] S. Zheng, K. Ristovski, A. Farahat, and C. Gupta, "Long short-term memory network for remaining useful life estimation," in *Proc. IEEE Int. Conf. Prognostics Health Manage. (ICPHM)*, Jun. 2017, pp. 88–95.

[23] J. W. Song, Y. I. Park, J.-J. Hong, S.-G. Kim, and S.-J. Kang, "Attention-based bidirectional LSTM-CNN model for remaining useful life estimation," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, May 2021, pp. 1–5.

[24] Y. Zhang, Y. Xin, Z.-W. Liu, M. Chi, and G. Ma, "Health status assessment and remaining useful life prediction of aero-engine based on BiGRU and MMoE," *Rel. Eng. Syst. Saf.*, vol. 220, Apr. 2022, Art. no. 108263.

[25] M. Ghifary, W. B. Kleijn, M. Zhang, D. Balduzzi, and W. Li, "Deep reconstruction-classification networks for unsupervised domain adaptation," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2016, pp. 597–613.

[26] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan, "Domain separation networks," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, 2016, pp. 343–351.

[27] Z. Du, J. Li, H. Su, L. Zhu, and K. Lu, "Cross-domain gradient discrepancy minimization for unsupervised domain adaptation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2021, pp. 3937–3946.

[28] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan, "A theory of learning from different domains," *Mach. Learn.*, vol. 79, nos. 1–2, pp. 151–175, May 2010.

[29] J. Li, E. Chen, Z. Ding, L. Zhu, K. Lu, and H. T. Shen, "Maximum density divergence for domain adaptation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 11, pp. 3918–3930, Nov. 2021.

[30] Y. Ganin *et al.*, "Domain-adversarial training of neural networks," *J. Mach. Learn. Res.*, vol. 17, no. 1, pp. 2030–2096, May 2015.

[31] J. Li, Z. Du, L. Zhu, Z. Ding, K. Lu, and H. T. Shen, "Divergence-agnostic unsupervised domain adaptation by adversarial attacks," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Sep. 3, 2021, doi: 10.1109/TPAMI.2021.3109287.

[32] E. Tzeng, J. Hoffman, N. Zhang, K. Saenko, and T. Darrell, "Deep domain confusion: Maximizing for domain invariance," 2014, *arXiv:1412.3474*.

[33] J. Li, M. Jing, H. Su, K. Lu, L. Zhu, and H. T. Shen, "Faster domain adaptation networks," *IEEE Trans. Knowl. Data Eng.*, early access, Feb. 19, 2021, doi: 10.1109/TKDE.2021.3060473.

[34] P. R. D. O. da Costa, A. Akçay, Y. Zhang, and U. Kaymak, "Remaining useful lifetime prediction via deep domain adaptation," *Rel. Eng. Syst. Saf.*, vol. 195, Mar. 2020, Art. no. 106682.

[35] C.-G. Huang, J. Zhu, Y. Han, and W. Peng, "A novel Bayesian deep dual network with unsupervised domain adaptation for transfer fault prognosis across different machines," *IEEE Sensors J.*, vol. 22, no. 8, pp. 7855–7867, Apr. 2022.

[36] J. Zhang, Y. Jiang, X. Li, M. Huo, H. Luo, and S. Yin, "An adaptive remaining useful life prediction approach for single battery with unlabeled small sample data and parameter uncertainty," *Rel. Eng. Syst. Saf.*, vol. 222, Jun. 2022, Art. no. 108357.

[37] F. O. Heimes, "Recurrent neural networks for remaining useful life estimation," in *Proc. Int. Conf. Prognostics Health Manage.*, Oct. 2008, pp. 1–6.

[38] J. Chen, H. Jing, Y. Chang, and Q. Liu, "Gated recurrent unit based recurrent neural network for remaining useful life prediction of nonlinear deterioration process," *Rel. Eng. Syst. Saf.*, vol. 185, pp. 372–382, May 2019.

[39] A. Saxena, K. Goebel, D. Simon, and N. Eklund, "Damage propagation modeling for aircraft engine run-to-failure simulation," in *Proc. Int. Conf. Prognostics Health Manage.*, Oct. 2008, pp. 1–9.

[40] W. Yu, I. Y. Kim, and C. Mechefske, "Remaining useful life estimation using a bidirectional recurrent neural network based autoencoder scheme," *Mech. Syst. Signal Proc.*, vol. 129, pp. 764–780, Aug. 2019.

[41] X. Li, Q. Ding, and J.-Q. Sun, "Remaining useful life estimation in prognostics using deep convolution neural networks," *Rel. Eng. Syst. Saf.*, vol. 172, pp. 1–11, Apr. 2018.

[42] M. Ragab *et al.*, "Contrastive adversarial domain adaptation for machine remaining useful life prediction," *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5239–5249, Aug. 2021.

[43] J. Li, X. Li, and D. He, "Domain adaptation remaining useful life prediction method based on AdaBN-DCNN," in *Proc. Prognostics Syst. Health Manage. Conf. (PHM-Qingdao)*, Oct. 2019, pp. 1–6.

[44] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial discriminative domain adaptation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jul. 2017, pp. 7167–7176.

[45] L. Van der Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. 11, pp. 2579–2605, Nov. 2008.

[46] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1050–1059.

Xinyao Li is currently pursuing the bachelor's degree with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China.

His research interests include transfer learning, machine learning, and prognostic health management.

Jingjing Li is currently a Professor with the University of Electronic Science and Technology of China (UESTC), Chengdu, China. His research interests include machine learning and multimedia analysis. Specifically, he focuses on domain adaptation, zero-shot learning, and recommender systems.

Lin Zuo is currently a Professor with the University of Electronic Science and Technology of China (UESTC), Chengdu, China. Her research interests include transfer learning and prognostic health management.

Lei Zhu (Senior Member, IEEE) is currently a Professor with Shandong Normal University, Shandong, China. His research interests include information retrieval and multimedia analysis.

Heng Tao Shen (Fellow, IEEE) is currently the Dean of the School of Computer Science and Engineering and the Executive Dean of the AI Research Institute, University of Electronic Science and Technology of China (UESTC), Chengdu, China. His research interests include multimedia search, computer vision, artificial intelligence, and big data management.

Dr. Shen is a fellow of the Association for Computing Machinery (ACM) and the Optical Society of America (OSA).