entropy

Article

PONDER: Enabling Balloon-Borne Based Solar Unmanned Aerial Vehicle's Take Off Diagnosis under Little Data

Yanfei Hu [1,†], Yingkui Jiao [2], Yujie Shang [3], Shuailou Li [1,†] and Yanpeng Hu [4,*]

1 Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100049, China; huyanfei@iie.ac.cn (Y.H.); lishuailou@iie.ac.cn (S.L.)
2 State Key Laboratory of Precision Measuring Technology and Instruments, Tianjin University, Tianjin 300072, China; jiaoyingkui@tju.edu.cn
3 School of Physics and Electrical Engineering, Anyang Normal University, Anyang 455008, China; asyj0606@163.com
4 School of Automation and Electrical Engineering, University of Science and Technology Beijing, Beijing 100083, China
* Correspondence: huyanpeng@ustb.edu.cn
† Current address: School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China.

Abstract: Balloon-borne based solar unmanned aerial vehicle (short for BS-UAV) has been researched prevalently due to the promising application area of near-space (i.e., 20–100 km above the ground) and the advantages of taking off. However, BS-UAV encounters serious fault in its taking off phase. The fault in taking off hinders the development of BS-UAV and causes great loss to human property. Thus, timely diagnosing the running state of BS-UAV in taking off phase is of great importance. Unfortunately, due to lack of fault data in the taking off phase, timely diagnosing the running state becomes a key challenge. In this paper, we propose PONDER to diagnose the running state of BS-UAV in the taking off phase. The key idea of PONDER is to take full advantage of existing data and complement fault data first and then diagnose current states. First, we compress existing data into a low-dimensional space. Then, we cluster the low-dimensional data into normal and outlier clusters. Third, we generate fault data with different aggression at different clusters. Finally, we diagnose fault state for each sampling at the taking off phase. With three datasets collected on real-world flying at different times, we show that PONDER outperforms existing diagnosing methods. In addition, we demonstrate PONDER's effectiveness over time. We also show the comparable overhead.

Keywords: unmanned aerial vehicle; fault diagnosis; deep learning; generative adversarial network; little data

check for updates

Citation: Hu, Y.; Jiao, Y.; Shang, Y.; Li, S.; Hu, Y. PONDER: Enabling Balloon-Borne Based Solar Unmanned Aerial Vehicle's Take Off Diagnosis under Little Data. *Entropy* **2022**, *24*, 997. https://doi.org/10.3390/e24070997

Academic Editor: António M. Lopes

Received: 10 June 2022
Accepted: 12 July 2022
Published: 19 July 2022

1. Introduction

Near-space (i.e., 20–100 km above the ground) vehicles have been becoming the focus of research and developing in the world due to the strategic significance (e.g., dealing with various tasks such as high-resolution earth observation, communication relay, atmospheric research, and so on) in numerous fields and great flight condition (e.g., low atmospheric density and strong solar radiation) [1–3]. Low- speed near-space vehicles utilize conventional taxiing and taking off mode, thus it has to consume more fuel and take a longer time to climb to the desired height. This makes the research on near-space vehicles full of challenges, especially on power supply and structure design [4]. To overcome these challenges, rather than directly focusing on the power supply or structure design, balloon-borne based solar powered unmanned aerial vehicles have begun to be studied in the aviation field for its special taking off [3].

Different from traditional takeoff from runway and landing, BS-UAV's take-off that is done with the help of a balloon's rise up entails the cooperation of multiple phases and components. The concrete process of a balloon's launch and take-off of BS-UAV is the following. The ball-borne based solar powered UAV is mounted by a high altitude balloon (the pitching angle of UAV is $-90°$ at this moment), and when the balloon rises up to the altitude of near space, then aerostation releases the solar powered UAV. The UAV implements longitudinal pull-up control and enters the cruise flight stage, and is recovered by sliding and landing. The state of each component in each phase is of importance for the success of taking off for BS-UAV.

Even though the great advantages for taking off compared to traditional near-space UAV, BS-UAV encounters more fault due to (multiple components' involvement in) its special takeoff mode and (light-level) structure design. The failure, especially before the pull-up phase, may lead to the crash of BS-UAV and a large cost on human property. Thus, timely diagnosing the overall state (e.g., normal or fault) by diving into the state of each component involved in the taking off of BS-UAV is of great importance. However, determining the overall state is challenging due to the characteristic of its taking off. First, the components involved in taking off process are diverse; as a result, there are numerous component faults or not an obvious fault for a component [5]. Second, the strong coupling relationship among each component poses a huge challenge for the overall diagnosing for BS-UAV's taking off, and the multiple components further aggravate such coupling. This makes it time-expensive for technicians to analyze the fault source accurately real-time; especially, the failure on taking off may be caused by different components each time. Third, the data information for each component is large-scale due to continuous monitoring [6]. These all hinder the fast fault identification.

To overcome the above challenge, many techniques exist. At a high level, these techniques could be classified into three categories. The first is rule-based diagnosis. It diagnoses a fault depending on a rule base among each component; however, such kinds of methods become ineffective as the design or the architecture is more and more complex. As a result, a machine learning-based fault diagnosis method that learns the pattern hidden behind multiple data for diagnosing faults is proposed [7,8]. The learning process of a machine learning-based diagnosis model requires feature selection from expertise [3]. As a diagnosing model learns feature data's distributions, different features that are utilized for a model lead to different diagnosing models. Thus, the human interference in feature selection is easy to cripple machine learning-based fault identification models. The third type is a deep learning-based diagnosis model [9–12]. Different from the human interference in feature selection, deep learning-based models [11] automatically learn a hidden representation on raw features based on predefined metrics and improve diagnosis accuracy. Deep learning-based models have presented multiple advantages in a fault identification domain.

Based on the above analysis, one feasible direction to accurately identify a fault in BS-UAV might be deep learning-based diagnosis models. The key idea of such kind of technique is to learn a model on the training data and use it to diagnose the fault in a real-time system once inputting a testing sample. The learning process of an effective diagnosis model requires plenty of data. However, actually, the data especially in a fault state are seldom due to the failure of taking off and thus the stop running or crash of each component. The lack of enough data in a fault state poses a critical challenge for applying such technique to timely determining the success of taking off or not.

In this paper, we propose PONDER, a deep learning-based fault diagnosis model combined with data complementation. More specially, PONDER complements fault data with a special generative network. This addresses the problem of a small amount of data in the fault state and allows us to diagnose the faults with a deep learning method in BS-UAV automatically. Deep learning learns a diagnosis model based on hidden representations that learns automatically from augmented training data, which allows us to diagnose fault real-time. We evaluated PONDER on the flying dataset, the evaluation results show that PONDER can identify the fault in BS-UAV's taking off effectively. In addition, PONDER can effectively diagnose a fault in UAV as time goes by compared with the state-of-the-art diagnosing methods. Finally, we measure the overhead it introduces. To conclude, we make the following contributions:

(1) We introduce a novel data augmentation method to solve the problem of data unbalancing in fault diagnosing. This allows us to diagnose fault accurately in UAV with little fault data.

(2) We design a diagnosing model that combines data generating and a deep learning technique to diagnose faults in BS-UAV.

(3) We evaluate the effectiveness of PONDER on real-world flying experiment dataset. We show that PONDER significantly improves the performance of an existing fault diagnosing model.

The rest of the paper is organized as follows: Section 2 is the background. Section 3 analyzes our insight of data generation technique. Section 4 gives the overview and the details of our proposed PONDER. Section 5 is the experimental set and evaluation. Section 6 describes the discussion. Related works and conclusions are given in Sections 7 and 8, respectively.

2. Background

In this section, we mainly discuss the background of deep learning based diagnosing method in mechanical field [13–20]. More details about BS-UAV (e.g., structure design or taking off manner) are shown in these works [1–3].

Research on fault diagnosis based on deep learning mainly includes Deep Belief Nets [9,21–23], Stacked Auto-Encoders [10], Recurrent Neural Networks [11], and Convolution Neural Networks [12]. DBN is a probabilistic generation model with a wide range of applications. It consists of multiple Restricted Boltzmann Machines layers, and there are connections between the layers but no connections within the layers. Training is divided into unsupervised pre-training and supervised back propagation fine-tuning for each layer. DBN is mainly used for feature extraction [24–28] and classification [29,30] in fault diagnosis. SAE is a multi-layer perceptual neural network composed of multiple self-encoding module units, which restores the original input to the maximum extent through the encoding and decoding process, and the model learns useful features by minimizing the differences between input and output, the model parameters are updated by the gradient descent algorithm, and the main functions of SAE are noise reduction filtering and feature extraction [3].

The processing units of RNN have both internal feedback and feed-forward connections, taking into account the correlation between samples, and can be used to process time series data or related data. RNN is often used in fault diagnosis for real-time fault diagnosis of complex industrial systems [9,11]. CNN is a feature extraction network composed of a convolutional layer and pooling layer alternately stacked, and then output by classification through the fully connection layer. It has the characteristics of sparse connection and weight sharing. Multiple convolution kernels perform convolution operations on the input to generate a convolution feature map. It is used for local feature extraction, combining abstraction layer by layer, and image recognition. CNN is very suitable for processing massive data [30,31], and fault diagnosis based on a CNN algorithm mainly uses CNN as feature extraction and recognition [32] or a classifier [25,29]. Most of these works focus on concrete design of a network structure but not the data challenges faced by them.

3. Insight of PONDER

Before demonstrating our method, we first introduce the insight of PONDER. As we know, the key aspect in PONDER is fault data generating. Thus, we focus on the insight of a generative adversary network. Indeed, the fault data are usually scattering around those normal data. We observe that the fault data for each component in the BS-UAV have activation bounds and cannot illustrate a meaningless value. Thus, we hold the view that the generated fault data should be dense around existing faults and sparse around normal data rather than arbitrarily scattering outside of the distribution of normal data. Based on this insight, we aim at generating fault data well aware. The key idea is to separate data of BS-UAV in the normal and fault state at first, and then generate more fault data around fault clusters and less around normal clusters. Considering that the data produced by each component are high-dimensional and such high-dimensional data make it difficult for clustering, we aim to compress high-dimensional data into a low-dimensional space.

4. Methodology of PONDER

We design PONDER to assist fault diagnosis in BS-UAV under the condition of unbalanced data. PONDER is short for "data comPlementatiON with Deep lEaRning based fault diagnosis model". In the following, we first explain the overview of PONDER, followed by the technical details of each component.

4.1. Overview of PONDER

We design the whole pipeline of PONDER, as shown in Figure 1. At a high level, PONDER mainly consists of three steps. The first step is pre-processing. In this step, we transform high-dimensional data into a compressed representation. The low-dimensional data allow us to cluster data in normal state and fault state accurately. In step2, we cluster on the compressed representation for separating different state's data, which make it easy for generating fault data well aware. In step3, we generate fault data utilizing a generative network, which allows us to discriminate fault state accurately and timely.

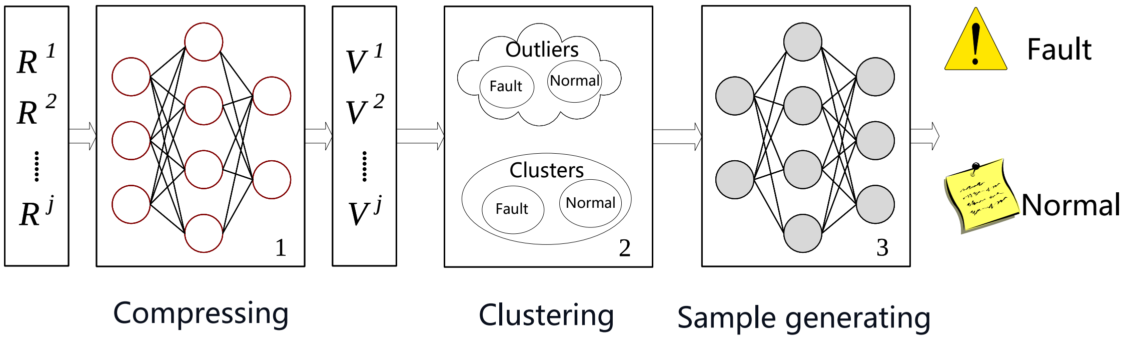

Figure 1. The overview of PONDER. PONDER consists of three modules. The first two are compressing network and clustering model, and the last is a model for fault data generating and fault diagnosing. R^j and V^j are the corresponding high dimensional raw data and low-dimensional representations.

Pre-processing. We transform high-dimensional data into low-dimensional representation. The low dimensional data allows us to perform accurate clustering. Given high dimensional data, we train an auto-encoder for compressing. Auto-encoder is a model with encoder and decoder. Encoder compresses high-dimensional data into low-dimensional representation, and decoder reconstructs high dimensional data based on the representation. More specifically, we train an auto-encoder with the goal that the high-dimensional data that decoder reconstructs is close or similar to original data. In this way, low-dimensional representation captures the corresponding high-dimensional features well and can be used for further clustering.

Clustering. We cluster data in normal states and that in fault conditions. The separation allows us to easily generate data around their distributions. Given the low-dimensional representations, we use a density-based clustering model called DBSCAN, and cluster existing representations into clusters. We use density-based clustering method with the following considerations. Density-based clustering method clusters data into normal state and fault state with the tolerance to abnormal data in dataset. These abnormal data form an outlier dataset and can be used as the reference for fault data generating. We do not utilize a K-means method since its clustering is sensitive to abnormal data. In addition, it is a distance-based clustering method and does not conform to our insights. In this step, we further cluster the outlier into a normal cluster and fault cluster.

Data Generating. We generate fault data depending on the clusters separated for further deep learning based fault identification. Given each cluster and the outlier, we train a modified generative adversarial network as our generating basis. Since we generate fault data based on clusters and outliers rather than learning existing distributions, this makes it different from the standard GAN. In this training, the generative adversarial network should be trained with the following rules. First, in areas near the cluster of fault data, we carefully expand the region of data and the expansion becomes less aggressive closer to data in normal conditions. Second, in the area of outliers, we obey the same rule based on an assumption that the outliers also obey each cluster's distributions.

Therefore, the GAN we used has two generators due to the presence of outliers and clusters. In addition, a discriminator in the GAN network is needed to discriminate the generated data and that of original data. We will introduce the details of each component in the following.

4.2. Technical Details

In this section, we present the technical details of each component in PONDER. We start by key notations. Given an input dataset X, Y, where X denotes the set of normal samples and fault samples, Y corresponds to their labels. Within the dataset, each sample $x \in R^{1\times p}$ is a p-dimensional vector, and the sample's label is represented by an integer value, indicating the corresponding state, i.e., $Y \in \{0, 1\}$, where 0 indicates data in normal states and 1 denotes data in fault conditions.

4.2.1. Auto-Encoder

Auto-encoder aims to transform high-dimensional data into low-dimensional representations to better identify the underlying clusterings. Based on the data in normal states, we want to learn an auto-encoder f to map any high-dimensional data x from X into h in a hidden space. As introduced in the overview part, the hidden space should maintain an accurate representation of raw data in high-dimension space. Thus, the loss for training our auto-encoder should be the following. Formally,

$$L_{auto} = \sum_{i=0}^{K} l_f(y_i, \overline{y_i}), i \in [0, K] \tag{1}$$

where $\overline{y_i}$ and y_i are the high-dimension data output by auto-encoder and the original high-dimensional data in the dataset, respectively. We utilize MSE as our loss function l_f, and train our auto-encoder with the goal of minimizing the loss, L_{auto}.

Note that here we do not use fault data for the auto-encoder's learning so that our auto-encoder may make the low-dimensional representation of fault data more separated. In the auto-encoder, the encoder's network structure is a Multi-layer Perceptron (MLP) with multiple hidden layers, and the decoder is a reverse of the encoder. With auto-encoder, we transform the feature in original feature space into a compact representation h in a hidden space H.

4.2.2. Dbscan

DBSCAN aims at clustering representations into normal and fault clusters for generating data based on them in the subsequent task. Concretely, first, it classifies representations into three categories with two parameters, d_r, t_{no}. d_r is a distance semidiameter, and t_{no} is the minimal number in the area of distance radius. The three categories of data point are core point, bound point, and outlier point. Then, DBSCAN classifies core point into different clusters. More concretely, it selects a core data point and assigns a cluster, and searches all the density connected points of this selected core point. It loops the selecting and assigning until the last point. Distance function is also a key point in determining the type of the data point, since DBSCAN needs to calculate d_r. We use L_2 norm as a distance function, i.e., Euclidean distance. We use a Silhoutte Coefficient as the clustering evaluation metric. Formally,

$$s(i) = b(i) - a(i)/max\{b(i), a(i)\}, i \in [1, K] \tag{2}$$

where $a(i)$ is an average distance of sample i to the other samples in same cluster, and $b(i)$ is an average distance of this sample to other samples in other clusters. Obviously, if the score of $s(i)$ is close to 1, it illustrates that the sample i has been well classified, and 0 means it might be a bound point between each cluster, -1 denotes it is a sample of other clusters.

Finally, we cluster data h including normal state h_n and fault state h_f into clusters h_c(h_{nc} and h_{fc}) and outliers h_o(h_{no} and h_{fo}). Note that we choose the value of t_{no} based on the rule of thumb that the value of t_{no} is similar with the order of the dimension. For our case, we have data of 10 dimensions; thus, t_{no} is 10.

4.2.3. Modified Generative Adversarial Network

The network aims to generate fault data based on each cluster output by DBSCAN. Based on each cluster, we want to learn a modified generative adversarial network (MGAN) for completing fault data generating. To this end, we should determine the network structure first and then explore the loss for training such network.

For determining the network's structure, as illustrated in the overview, after clustering, the data have been classified into two clusters and two outliers. Thus, the fault data that need to be generated have two types. The first type is a distribution that conforms to the clustered distribution, and the second is a distribution that fits to the outlier area. Based on this analysis, the concrete structure of the modified generative adversarial network is shown in Figure 2.

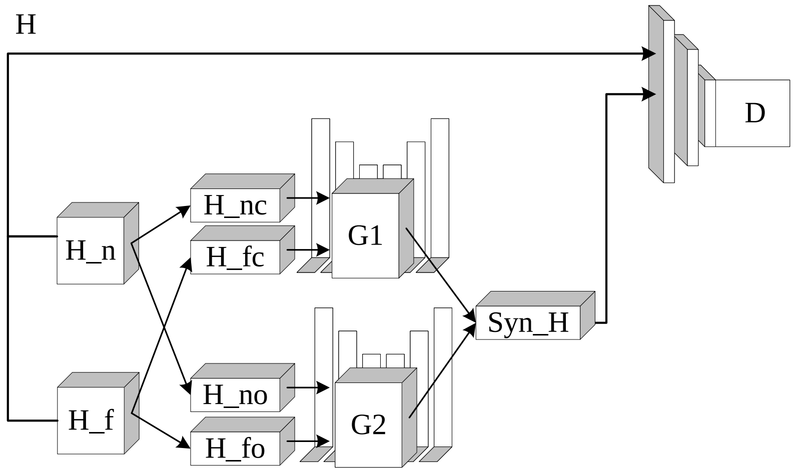

Figure 2. The details of MGAN. *H_n* and *H_f* are representations of normal state and fault state in hidden space. *c* and *o* denote clusters and outliers. *G* and *D* represent generator and discriminator. *Syn_H* means the synthetic data of fault state.

Next, we explore the loss for the two generators. To make generative network learn the real distribution of fault data, when we learn the distribution in the clusters and outliers, we all expand the fault data region carefully around normal cluster or normal outlier regions. Specifically,

Data Synthesis in the Cluster Region. For learning the real distribution d_{rc} conforming to (approximating) the data in clusters [33], as shown in the upper part of Figure 2, we train G1 to simulate this distribution, where G1 generates data in high-density regions. Concretely, if the probability of the data $\tilde{s}$ generated by G1 falling into the high-density regions of normal states is bigger than a threshold $p_n(\tilde{s}) > \lambda$, it will be generated with a lower probability; otherwise, it will generate data with a distribution balancing clustered data of normal states and fault states. We use α to control the generated sample to approximate normal or fault clusters. We define the distribution required as the following:

$$d_c = \begin{cases} \alpha p_n(\tilde{s}) + (1-\alpha)p_f(\tilde{s}) & , p_n(\tilde{s}) \leq \lambda \wedge \tilde{s} \in H \\ \dfrac{1}{\tau p_n(\tilde{s})} & , p_n(\tilde{s}) > \lambda \wedge \tilde{s} \in H \end{cases} \tag{3}$$

where τ is the normalization term, and λ is a threshold to indicate whether the generated data are in high-density normal regions. H denotes the whole hidden space.

In order to learn this distribution d_c, we minimize the KL divergence between defined distribution that we will learn and real distribution of clusters d_{rc}. The function is the following:

$$L_{KL(d_{rc}||d_c)} = -\mathcal{H}(d_{rc}) + \mathop{\mathrm{E}}_{\tilde{s}\sim d_{rc}} \log P_{d_c}(\tilde{s}) \tag{4}$$

KL divergence is the expectation value of the logarithmic difference of d_{rc} and d_c on d_{rc} distribution. After transformation, it can be divided into two parts. The first term is the opposite value of d_{rc} entropy, and the second term is the expectation of $\log P_{d_c}(\tilde{s})$ on the d_{rc} distribution. $\log P_{d_c}(\tilde{s})$ means the logarithmic of the probability of sample in d_c distribution.

In addition, we should make the generated data in a fault condition and the real fault data indistinguishable. In other words, the generated data are located in a clustered region as far as possible. Thus, L_{clus} is required as the following:

$$L_{clus} = || \mathop{\mathrm{E}}_{\tilde{s}\sim d_{rc}} f(\tilde{s}) - \mathop{\mathrm{E}}_{s\sim h_c} f(s)||^2 \tag{5}$$

Here, f is the function of discriminator.

Overall, the loss for learning generator $G1$ is

$$L_{G1} = L_{KL(d_{rc}||d_c)} + L_{clus} \tag{6}$$

Data Synthesis at Outlier Region. As we have clustered the outliers into two clusters, the loss for second generator is the same as that of L_{G1}.

Discriminator. Different from the commonly used generative adversarial network that aims at discriminating synthesized fault data from real fault data, the special designed discriminator aims to discriminate normal data from fault data. Concretely, it discriminates normal data from synthesized fault data, and from real fault data. The discriminator not only participates in the training process but also acts as the diagnoser in diagnosing steps. For learning this discriminator, we aim to optimize the following loss function:

$$\begin{aligned} L_{disc} = &\mathop{\mathrm{E}}_{s\sim p_n}[\log D(s)] + \mathop{\mathrm{E}}_{\tilde{s}\sim d_{rc}}[\log(1 - D(\tilde{s}))] + \\ &\mathop{\mathrm{E}}_{\tilde{s}\sim d_{ro}}[\log(1 - D(\tilde{s}))] + \mathop{\mathrm{E}}_{s\sim p_n}[D(s)\log D(s)] + \\ &\mathop{\mathrm{E}}_{s\sim p_f}[\log(1 - D(s))] \end{aligned} \tag{7}$$

Here, the first three terms distinguish normal data from synthesized fault data. The fourth conditional entropy term is to recognize normal data with high confidence, which corresponds with our assumption. The last term encourages the discriminator to correctly classify normal data from real fault data. With all the terms, the discriminator could be learned to classify data in normal states from both real and synthesized fault data.

4.2.4. Hyper-Parameters

The batch size is 128, and the training epoch is 250. We use the Adam optimizer for minimizing the loss of L_{auto}, L_{G1} L_{G2} and L_{disc}. The technical details of this optimization technique can refer to [34]. Here, the batch size and training epoch can be randomly selected. The larger the batch size, the better parameter learning. Note that we should control batch size value to avoid out of memory in training.

For auto-encoder, the encoder has two hidden layers. The input size has dimensions of 60, and the output size is 10 dimensions. We set the dimension size of two hidden layers as 30 and 20 dimensions, respectively. For MGAN, the discriminator and the generators are feed-forward neural networks. Each network of generators and discriminator contains hidden layers (with dimensions of 20 and 5, respectively). The dimension of noise for each generator is 40, and the output dimension of generators is the same as that of the auto-encoder, which is 10. The threshold λ is set as 0.95 of the probability of normal data predicted by a pre-trained probability estimator. We set τ to a small value 0.1. The last two settings of hyper-parameters (i.e., λ and τ) have already been shown their effectiveness in this work [33]. We use a pull-away term to minimize $H(d_{rc})$ [35] and the technique proposed by [36] that uses a neural network to approximate p_n. For the hyper-parameters, we use these default parameters and do not consider the sensitivity of them to model performance.

5. Evaluation

We evaluate the effectiveness of PONDER on fault diagnosis with the flying data of BS-UAV. The evaluation are performed on a server with two Intel Xeon E5-2630 v3 2.4 GHz CPUs (32 processors) and 128 GB memory. We focus on four key aspects: (a) design choice, (b) effectiveness, (c) robustness, and (d) overhead.

5.1. Experimental Setup

Datasets. We prepare three different datasets for evaluating PONDER in our experiment. Each dataset is made up of flying data of the same type of BS-UAV in normal conditions and fault condition, with 354,000 normal items and 6000 fault items (little fault data). Each sample is a matrix of 1×60. The time interval of three datasets is over half a year, which enables us to evaluate the robustness of PONDER over time. More specifically, they are collected at May 2019, February 2020, and October 2020. For convenience, we refer them as D-May, D-Feb, and D-Oct, respectively.

For each dataset, D-May is mainly used for training and testing. D-Feb and D-Oct are used for testing. More specifically, we randomly divide D-May into two parts, 70% of D-May is training dataset (Training), and the other is testing dataset (Testing). In addition, 20% of the samples in the training dataset are held out for validation.

Baseline Models. We utilize three existing methods as our baseline models. More specifically, for validating design choices, we assess the performance of PONDER with a deep learning based diagnosing model with one generator; their difference only exists in the number of generators. For evaluating PONDER's performance, we compared PONDER with the state-of-the-art data generation techniques, ODDS [33]. ODDS works under the same situation as a few samples as ours. In addition, we also compare PONDER with machine learning based models that use Random Forest (RF) as the diagnosing algorithm. RF is a model with multiple groups of parameters or models and behaves best among machine learning based models. These baseline models allow us to evaluate PONDER thoroughly.

Metrics. As the essential of the fault diagnosing is a classification problem, we choose metrics widely used in a classification domain. These metrics include precision (the correct diagnosis among all the diagnosis determined by diagnosis model), recall (the true parts among the actual ones), and F1-score. F1-score indicates a comprehensive evaluation for the performance, which balances precision and recall: $F1 = 2 \times Precision \times Recall/(Precision + Recall)$. In addition, we also utilize false positive rate (FPR) that equals $1 - recall$ as our evaluation metrics. We choose them (e.g., precision, recall, F1-score and FPR) as the final measurement for the performance of each diagnosing model if not otherwise stated.

Training and Testing. Generally, we use training dataset for learning each model's parameters, and use validation dataset for deciding their best hyper-parameters. We test each model on a testing dataset and evaluate each model's performance by calculating each metric based on a testing dataset.

Specifically, for building PONDER, first, we use all normal data in dataset Training and learn the parameters of auto-encoder, and perform dimension compressing on each sample in dataset Training by using a learned auto-encoder. Then, we cluster on all low-dimensional representations in dataset Training into four clusters. The four clusters include two clusters of normal data and fault data, and two clusters of outliers of normal data and fault data. Third, we train the generator based on normal clusters and outliers. For ODDS, the building procedure is the same except the lack of clustering for outlier samples. For RF, we directly train them on dataset Training.

For testing PONDER, in other words, diagnosing if there exists a fault using PONDER, we import each sample of testing dataset into the above learned generator. The outputs of generator are the diagnosed result. Fault diagnosis using ODDS and RF are the same steps except for importing each sample into their models.

Implementation of PONDER. Following each subsection including hyper-parameters of Section 4.2, the network details (e.g., neural network's layer number and types) can be obtained, and the network model (with network parameters) could be trained by minimizing four losses (i.e., L_{auto}, L_{G1}, L_{G2}, and L_{dis}) with the Adam optimizer. We show the work-flow of building PONDER and that of applying PONDER to fault diagnosing in Figure 3.

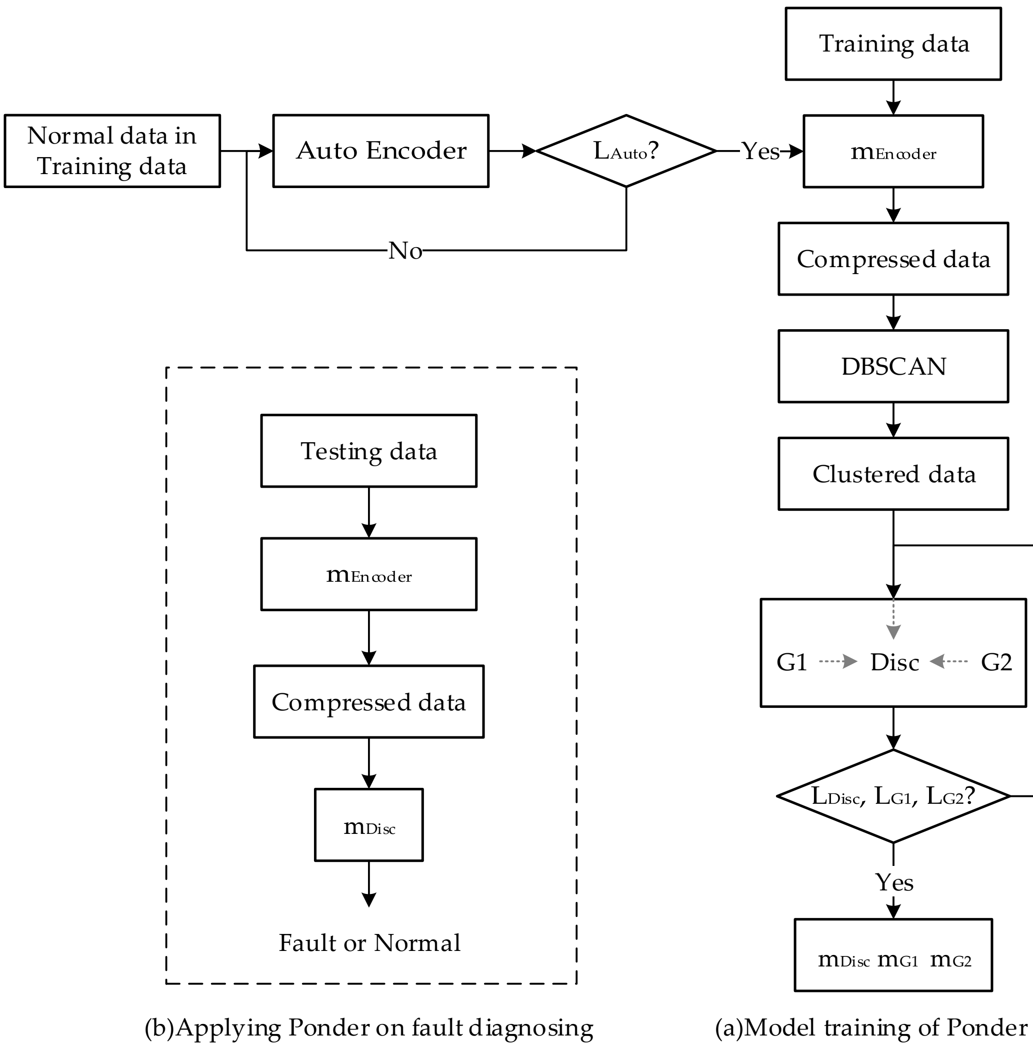

Figure 3. The work-flow of PONDER's building and diagnosing.

Detailed Evaluation. Based on the above setup, we aim to further refine the evaluation aspects into the following four questions:

$Q1_a$: How is the performance of PONDER compared to that of DMO under the circumstance of little data?

$Q2_b$: How is the performance of PONDER compared to existing diagnosing models?

$Q3_c$: How does PONDER behave over time?

$Q4_d$: How much overhead does the PONDER introduce to the diagnosing model?

5.2. Experimental Design and Results

In this section, we aim to answer each question in detail. For each question, we will first describe the experiment design if necessary, then the results.

$Q1_a$: How is the performance of PONDER compared to that of DMO under the circumstance of little data?

To validate the design choice of two generators, we compare the performance of PONDERwith that of only one generator. Their results are shown in Table 1. At a high level, we can see that PONDER outperforms that of one generator on two datasets. Concretely, the F1-score of PONDER on the test dataset arrives at 0.950, while models that generate fault data with only one generator have the F1-score of 0.931 and 0.927. The performance on D-Feb. has the same conclusion and is omitted for brevity. These results all indicate that PONDER's two generators are better than that with one generator for fault data generating.

Table 1. F1-score comparison of PONDER models with different numbers of generators.

Dataset	G1 Generator	G2 Generator	Both Generators
Testing	0.931	0.927	0.950
D-Feb.	0.917	0.911	0.932

$Q2_b$: How is the performance of PONDER compared to existing diagnosing models?

We compare the performance of PONDER, RF, and ODDS on dataset D-Feb to evaluate the performance of PONDER. Their results are shown in Table 2. We can see that PONDER outperforms all the baseline models. More specially, PONDER has an F1-score of 0.93, while RF has an F1-score of 0.83. The reason for RF's smaller results might be the lack of future distribution of fault data. ODDS detects fault samples with the F1-score of 0.91. The explanation for the lower F1-score of ODDS might be the assumption that the fault data in outlier are scattered randomly.

Table 2. Performance comparison of PONDER models and baseline models on two datasets.

Dataset	**D-Feb**				**D-Oct**				
Metric	**Precision**	**Recall**	**F1-Score**	**FPR**	**Prec.**	**Rec.**	**F1.**	**FPR**	**Time (ms)**
RF	0.82	0.84	0.83	0.08	0.78	0.79	0.79	0.08	0.17
ODDS	0.90	0.92	0.91	0.06	0.88	0.87	0.87	0.06	0.11
PONDER	0.92	0.94	**0.93**	0.04	0.90	0.91	**0.91**	0.04	0.12

In addition, we also notice that PONDER diagnoses fault with the lowest false positive rate. The lowest FPR means that there are less false diagnoses where normal state is regarded as false state. These results all illustrate that PONDER takes the characteristic of fault data and helps to improve the performance of existing fault diagnosing methods.

$Q3_c$: How does PONDER behave over time?

To evaluate the PONDER's robustness over time, we compare the performance of PONDER on D-Feb and D-Oct. In addition, we also give the results of RF and ODDS as a comparison. Their results are shown in Table 2. We can see that PONDER does not have obvious degradation on D-Oct. However, RF and ODDS detect fault data with little decline. More specifically, the F1-score of PONDERdeclines by 2%, and the baseline models of RF and ODDS decline by 4%.

As we do not have more data to validate the degradation over time, we verify that, on these three datasets, PONDER behaves better over time. This demonstrates that PONDER is robust as time goes by.

$Q4_d$: How much overhead does the PONDER introduce to the diagnosing model?

We record the average fault detection time that PONDER spends on data processing and fault detection for each sample in D-Oct. We do not consider the time spent on training each model, since these models are trained offline and the training time could be compensated by a large amount of data. As a comparison, we also give the results of a baseline model. The detecting time of each model is shown in the last column of Table 2. We can see that the average detection time of PONDER models is comparable to that of baseline models; in particular, it is more similar to that of the ODDS model.

6. Discussion

Static vs. Learning based Model. Static diagnosing has high precision but with expensive manual cost. In cases without strong coupling, static techniques such as domain-expertise or a rule based method is a better choice. For cases where strong coupling exists among components, learning based methods have superiority. PONDER is a kind of learning based diagnosing method for biased data.
Real-time Diagnosing. Even though PONDER is evaluated on the offline data, it can be used for real-time diagnosing. It is possible to transform software codes into hardware language and implemented it with hardware. We will leave the further hardware implementation to future work.
Limitations. PONDER works under an assumption that data in normal data are stable. However, actually normal data may change with time going, for example, a substituted component that has the same function with original but with different data expressing. Thus, PONDER needs to be updated for a certain of time. In addition, to determine when to update the diagnosing model, authors in [37] have introduced methods.

7. Related Works

Anomaly Diagnosis. Anomaly diagnosis techniques diagnose fault state with only normal data. They make an assumption that normal data are stable and unchanged for a period. In our work, we make the same assumption, while the difference is that we take full advantage of the little fault data. We do not use it as our baseline since it has been verified by these works [33].
Data Augmentation. Data augmentation is a technique derived from computer domains and usually used for complementing the needed data. It usually achieves its goal via a generative adversarial network. Current data augmentation works mainly use it to generate data based on existing data distribution or learn an unknown distribution with existing data distribution. The most related work to ours is [33]. In Ref. [33], the authors design two generators for assisting bot data generation based on two different assumptions. However, we still use two generators but with a different manner. We assume that the fault data that need to be generated conform to the same assumption (i.e., dense at fault clusters or outliers and sparse at normal clusters or outliers). In addition, the application domain is essentially different. We do not consider other related works in terms of auto-encoder and clustering, as the two techniques are just an assist for our data generation and have no difference from others.

8. Conclusions

This paper presents PONDER, a fault diagnosing model for BS-UAV's taking off with little data. By designing novel generators for learning fault data's distribution, we generate plenty of fault data and enable a deep learning based fault detection model to diagnose BS-UAV's take off timely.

Author Contributions: Y.H. (Yanpeng Hu) proposed the main the idea; Y.H. (Yanfei Hu) prepared the manuscript initially; Y.J., Y.S. and S.L. performed the experiment. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hu, Y.; Yang, Y.; Ma, X.; Li, S. Computational optimal launching control for balloon-borne solar-powered unmanned aerial vehicles in near-space. *Sci. Prog.* **2020**, *103*, 0036850419877755. [CrossRef] [PubMed]
2. Hu, Y.; Guo, J.; Meng, W.; Liu, G.; Xue, W. Longitudinal Control for Balloon-Borne Launched Solar Powered UAVs in Near-Space. *J. Syst. Sci. Complex.* **2022**, *35*, 802–819. [CrossRef]
3. Keizer, M.C.O.; Teunter, R.H. Clustering condition-based maintenance for a multi-unit system with aperiodic inspections. In *Safety and Reliability of Complex Engineered Systems: Proceedings of the 25th European Safety and Reliability Conference, ESREL 2015, Zürich, Switzerland, 7–10 September 2015*; CRC Press: Boca Raton, FL, USA, 2015; pp. 983–991.
4. Froger, A.; Gendreau, M.; Mendoza, J.E.; Pinson, E.; Rousseau, L.-M. Maintenance scheduling in the electricity industry: A literature review. *Eur. J. Oper. Res.* **2016**, *251*, 695–706. [CrossRef]
5. Li, W. Advance of intelligent fault diagnosis for complex system and its present situation. *Comput. Simulation* 2004, *21*, 4–7.
6. LeCun, Y.; Bengio, Y.; Hinton, G. Deep learning. *Nature* **2015**, *521*, 436–444. [CrossRef]
7. Han, L.; Deyun, X. Survey on data driven fault diagnosis methods. *Control. Decis.* **2011**, *26*, 1–9.
8. Lei, Y.; Jia, F.; Lin, J.; Xing, S.; Ding, S.X. An intelligent fault diagnosis method using unsupervised feature learning towards mechanical big data. *IEEE Trans. Ind. Electron.* **2016**, *63*, 3137–3147. [CrossRef]
9. Liu, R.; Yang, B.; Zio, E.; Chen, X. Artificial intelligence for fault diagnosis of rotating machinery: A review. *Mech. Syst. Signal Process.* **2018**, *108*, 33–47. [CrossRef]
10. Mao, W.; Feng, W.; Liang, X. A novel deep output kernel learning method for bearing fault structural diagnosis. *Mech. Syst. Signal Process.* **2019**, *117*, 293–318. [CrossRef]
11. Girshick, R.; Donahue, J.; Darrell, T.; Malik, J. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Columbus, OH, USA, 23–28 June 2014; pp. 580–587. [CrossRef]
12. Vincent, P.; Larochelle, H.; Lajoie, I.; Bengio, Y.; Manzagol, P.-A.; Bottou, L. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *J. Mach. Learn. Res.* **2010**, *11*, 3371–3408.
13. Gan, M.; Wang, C.; Zhu, C. Construction of hierarchical diagnosis network based on deep learning and its application in the fault pattern recognition of rolling element bearings. *Mech. Syst. Signal Process.* **2016**,*72*, 92–104. [CrossRef]
14. Jia, F.; Lei, Y.; Guo, L.; Lin, J.; Xing, S. A neural network constructed by deep learning technique and its application to intelligent fault diagnosis of machines. *Neurocomputing* **2018**, *272*, 619–628. [CrossRef]
15. Li, K.; Wang, Q. Study on signal recognition and diagnosis for spacecraft based on deep learning method. In Proceedings of the Prognostics and System Health Management Conference (PHM), Beijing, China, 21–23 October 2015; pp. 1–5.
16. Li, X.; Zhang, W.; Ding, Q. A robust intelligent fault diagnosis method for rolling element bearings based on deep distance metric learning. *Neurocomputing* **2018**, *310*, 77–95. [CrossRef]
17. Miao, Z.H.; Zhou, G.X.; Liu, H.N. Tests and feature extraction algorithm of vibration signals based on sparse coding. *J. Vib. Shock.* **2014**, *33*, 76–81.
18. Shao, H.; Jiang, H.; Lin, Y.; Li, X. A novel method for intelligent fault diagnosis of rolling bearings using ensemble deep auto-encoders. *Mech. Syst. Signal Process.* **2018**, *102*, 278–297. [CrossRef]
19. Shao, H.; Jiang, H.; Zhao, H.; Wang, F. A novel deep autoencoder feature learning method for rotating machinery fault diagnosis. *Mech. Syst. Signal Process.* **2017**, *95*, 187–204. [CrossRef]
20. Sun, W.; Shao, S.; Zhao, R.; Yan, R.; Zhang, X.; Chen, X. A sparse auto-encoder-based deep neural network approach for induction motor faults classification. *Measurement* **2016**, *89*, 171–178. [CrossRef]
21. Hinton, G.; Osindero, E.S.; Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural Comput.* **2006**, *18*, 1527–1554. [CrossRef]
22. Shao, H.; Jiang, H.; Zhang, X.; Niu, M. Rolling bearing fault diagnosis using an optimization deep belief network. *Meas. Sci. Technol.* **2015**, *26*, 115002. [CrossRef]
23. Tamilselvan, P.; Wang, P. Failure diagnosis using deep belief learning based health state classification. *Reliab. Eng. Syst. Saf.* **2013**, *115*, 124–135. [CrossRef]
24. Guo, X.; Chen, L.; Shen, C. Hierarchical adaptive deep convolution neural network and its application to bearing fault diagnosis. *Measurement* **2016**, *93*, 490–502. [CrossRef]
25. Janssens, O.; Slavkovikj, V.; Vervisch, B.; Stockman, K.; Loccufier, M.; Verstockt, S.; Van de Walle, R.; Van Hoecke, S. Convolutional Neural Network Based Fault Detection for Rotating Machinery. *J. Sound Vib.* **2016**, *377*, 331–345. [CrossRef]
26. Jeong, H.; Park, S.; Woo, S.; Lee, S. Rotating machinery diagnostics using deep learning on orbit plot images. *Procedia Manuf.* **2016**, *5*, 1107–1118. [CrossRef]
27. Jia, F.; Lei, Y.; Lin, J.; Zhou, X.; Lu, N. Deep neural networks: A promising tool for fault characteristic mining and intelligent diagnosis of rotating machinery with massive data. *Mech. Syst. Signal Process.* **2016**, *72–73*, 303–315. [CrossRef]

28. Wang, K.; Zhao, Y.; Xiong, Q.; Fan, M.; Sun, G.; Ma, L.; Liu, T. Research on healthy anomaly detection model based on deep learning from multiple time-series physiological signals. *Sci. Program.* **2016**, *2016*, 5642856. [CrossRef]
29. Guo, S.; Yang, T.; Gao, W.; Zhang, C. A Novel Fault Diagnosis Method for Rotating Machinery Based on a Convolutional Neural Network. *Sensors* **2018**, *18*, 1429. [CrossRef]
30. Jia, F.; Lei, Y.; Lu, N.; Xing, S. Deep normalized convolutional neural network for imbalanced fault classification of machinery and its understanding via visualization. *Mech. Syst. Signal Process.* **2018**, *110*, 349–367. [CrossRef]
31. Zhang, W.; Li, C.; Peng, G.; Chen, Y.; Zhang, Z. A deep convolutional neural network with new training methods for bearing fault diagnosis under noisy environment and different working load. *Mech. Syst. Signal Process.* **2018**, *100*, 439–453. [CrossRef]
32. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 26 June–1 July 2016; pp. 770–778.
33. Jan S.T.; Hao, Q.; Hu, T.; Pu, J.; Oswal, S.; Wang, G.; Viswanath, B. Throwing darts in the dark? detecting bots with limited data using neural data augmentation. In Proceedings of the 2020 IEEE symposium on security and privacy (SP), San Francisco, CA, USA, 18–21 May 2020; pp. 1190–1206.
34. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv* **2014**, arXiv:1412.6980.
35. Dai, Z.; Yang, Z.; Yang, F.; Cohen, W.W.; Salakhutdinov, R.R. Good semi-supervised learning that requires a bad gan. *Adv. Neural Inf. Process. Syst.* **2017**, *30*. Available online: https://www.xueshufan.com/publication/2963170156 (accessed on 1 December 2021).
36. Niculescu-Mizil, A.; Caruana, R. Predicting good probabilities with supervised learning. In Proceedings of the 22nd International Conference on Machine Learning, Bonn, Germany, 7–11 August 2005; pp. 625–632.
37. Yang, L.; Guo, W.; Hao, Q.; Ciptadi, A.; Ahmadzadeh, A.; Xing, X.; Wang, G. {CADE}: Detecting and explaining concept drift samples for security applications. In Proceedings of the 30th USENIX Security Symposium (USENIX Security 21), Virtual, 11–13 August 2021; pp. 2327–2344.