Proceedings of the ASME 2023 18th International Manufacturing Science and Engineering Conference MSEC2023 June 12 - June 16, 2023, New Brunswick, NJ, USA

MSEC2023-101281

NORMALIZING FLOWS FOR INTELLIGENT MANUFACTURING

Matthew Russell[1],*, Peng Wang[1]

[1]University of Kentucky, Lexington, KY

ABSTRACT

Monitoring machine health and product quality enables predictive maintenance that optimizes repairs to minimize factory downtime. Data-driven intelligent manufacturing often relies on probabilistic techniques with intractable distributions. For example, generative models of data distributions can balance fault classes with synthetic data, and sampling the posterior distribution of hidden model parameters enables prognosis of degradation trends. Normalizing flows can address these problems while avoiding the shortcomings of other generative Deep Learning (DL) models like Generative Adversarial Networks (GAN), Variational Autoencoders (VAE), and diffusion networks. To evaluate normalizing flows for manufacturing, experiments are conducted to synthesize surface defect images from an imbalanced data set and estimate parameters of a tool wear degradation model from limited observations. Results show that normalizing flows are an effective, multi-purpose DL architecture for solving these problems in manufacturing. Future work should explore normalizing flows for more complex degradation models and develop a framework for likelihood-based anomaly detection. Code is available at https://github.com/uky-aism/flows-for-manufacturing.

Keywords: Condition Monitoring, Deep Generative Models, Normalizing Flows, Parameter Estimation

1. INTRODUCTION

Predictive maintenance depends on monitoring machine health and product quality. Data-driven approaches rely on machine learning and Deep Learning (DL) to extract features from raw sensing data and predict health indicators and assess quality. The constraints of manufacturing applications motivate exploration of generative models. For example, the scarcity of data from abnormal health conditions hinders training DL models but can be alleviated by generating realistic fake data to expand the training set. Similarly, approximating the posterior distribution of degradation model parameters can estimate future degradation trends and Remaining Useful Life (RUL). However, both the data distribution and parameter distribution are usually too complex to compute analytically.

Fortunately, advances in DL have produced data-driven generative techniques that can synthesize data and estimate parameters by approximating complex distributions. Popular generative DL models include Generative Adversarial Networks (GAN) [1], Variational Autoencoders (VAE) [2, 3], and diffusion networks [4] (e.g., DALL-E 2 [5] and Stable Diffusion [6]). However, all three approaches encounter difficulties during training or inference. Alternatively, normalizing flows leverage an invertible random variable transformation that simplifies both training and inference. The core idea is to normalize a complex distribution (e.g., that of surface defect images) by transforming it into a tractable distribution like a standard normal distribution. By inverting the transformation, samples from the tractable distribution can be mapped back into the input domain to generate samples from the complex distribution.

Despite the potential advantages, very little work has been done with normalizing flows in manufacturing. Zhang et al. [7] used normalizing flows to model the distribution of bearing vibration signals for anomaly detection. The flow transformed the signals into a standard normal distribution where the vector norm was used for threshold-based, one-class anomaly detection. However, the results showed minimal improvements over simpler, root mean square (RMS)-based methods. Both Rudolph et al. [8] and Szarski and Chauhan [9] adopted similar flow-based anomaly detection methods for product defect images, but neither provided a strong case for using normalizing flows over alternative methods. In another direction, Yang et al. [10] noticed that flows can be used to normalize the distribution of latent codes in a bidirectional Gated Recurrent Unit (GRU) network, making it easier for a decoder to predict the RUL of turbofan jet engines. However, normalizing flows have considerably broader applications in manufacturing that remain unexplored, notably, synthetic data generation and parameter estimation.

Inspired by the general utility and multifaceted applications of normalizing flows, this study offers the following contributions:

*Corresponding author: matthew.russell@uky.edu

1. a theoretical overview to normalizing flows in the context of manufacturing, and

2. two case studies evaluating the use of normalizing flows for synthesizing surface defect images and estimating parameters of a tool wear degradation model.

In the remainder of this paper, Section 2 describes the motivation and theory behind normalizing flows, Section 3 discusses the proposed applications in machine condition monitoring, Sections 4 and 5 present two case studies, and Section 6 presents concluding remarks.

2. GENERATIVE MODELING WITH NORMALIZING FLOWS

Generative models can answer questions about data distributions. That is, a generative model of $p(x)$ can query the distribution of x for new examples or estimate the likelihood (density) of a given x.

2.1 Related Generative Methods

Related methods include Generative Adversarial Networks (GAN), Variational Autoencoders (VAE), and diffusion networks. GANs combine a discriminator network with an adversarial generator network that synthesizes new samples from the data distribution [1]. The discriminator tries to distinguish real data samples from fake examples produced by the generator, acting as a trainable metric for determining distribution membership. While the discriminator is trained to better distinguish real and fake data, the generator is trained to produce examples that confuse it, i.e., look like real data. GANs cannot estimate density and can be difficult to train since the competing networks must be properly balanced [11].

VAEs use a latent variable generative model in which a hidden code z generates the observed x [2, 3]. As a variational method, VAE attempts to infer a distribution of codes that matches a simple prior $q(z)$. To do this, a probabilistic encoder infers the parameters of $p(z \mid x)$ from a data point, and a probabilistic decoder predicts $p(x \mid z)$ using $z \sim p(z \mid x)$ to reconstruct the input. During training, the probabilistic encoder learns to encode key semantics about x into z while matching the prior $q(z)$ (e.g., a standard normal distribution). Then samples from $p(x)$ are drawn by first sampling the prior $q(z)$ and using the probabilistic decoder. Like GANs, VAEs cannot provide density estimates, and it can be difficult to balance learning a good reconstruction model $p(x \mid z)$ and regularizing the latent codes to match the desired $q(z)$ [12, 13].

Diffusion networks model the generative process as a multi-step reverse diffusion process [4]. The forward diffusion process progressively destroys the input through additive Gaussian noise. A neural network is trained to reverse the stochastic process and gradually materialize an image from noise. While producing impressive synthetic images [5, 6], these networks often require a lengthy and iterative reverse process to generate samples. The computational requirements make them less suitable for generating large batches of simulated data (e.g., synthetic defect images to balance a condition monitoring data set). Like GANs and VAEs, diffusion networks cannot produce density estimates. In contrast, normalizing flows offer an alternative that avoids many shortcomings of all three approaches.

2.2 Normalizing Flows

Let $p_\theta(x)$ be a generative model parameterized by θ that should match the true data distribution $p(x)$, i.e., θ should be chosen to minimize the KL Divergence between the two:

$$\begin{aligned} \theta^* &= \arg\min_\theta D_{\mathrm{KL}}(p(x) \,||\, p_\theta(x)) \\ &= \arg\min_\theta \mathbb{E}_{x\sim p(x)}\left[\log p(x)\right] - \mathbb{E}_{x\sim p(x)}\left[\log p_\theta(x)\right] \end{aligned} \tag{1}$$

Since the first term does not include θ, this minimization objective is identical to maximizing the second term, the model's log likelihood over the data set:

$$\theta^* = \arg\max_\theta \mathbb{E}_{x\sim p(x)}\left[\log p_\theta(x)\right] \tag{2}$$

Predicting $p_\theta(x)$ directly with a neural network would result in a trivial solution where all x map to infinite density and would not permit efficient sampling of $p_\theta(x)$.

To resolve this, normalizing flows implement an invertible random variable transformation (RVT) $f_\theta(x)$ with tractable Jacobian determinant $\det J$ that computes the log likelihood by mapping x to a base distribution $q(\epsilon)$:

$$\log p_\theta(x) = \log q(f_\theta(x)) + \log|\det J| \tag{3}$$

The determinant of the Jacobian itself captures how the transformation scales volume. A small value indicates that the transformation condenses data to a single point, and thus maximizing it avoids this outcome. Therefore, the two terms work in unison to pull data towards the maximum likelihood point of $q(\epsilon)$ while preventing the mapping from collapsing [14]. From Eq. (2) and Eq. (3), the normalizing flow optimization problem is then

$$\theta^* = \arg\max_\theta \frac{1}{N}\sum_{i=1}^{N}\left[\log q(f_\theta(x_i)) + \log|\det J|\right] \tag{4}$$

where $q(\epsilon)$ is usually a simple distribution like a standard normal or Bernoulli distribution, and the expectation is replaced with an approximation using samples x_i from the data set. If $f_\theta(\cdot)$ is a neural network, the training loss function is:

$$\mathscr{L}_{\mathrm{NF}}(x_i) = -\log q(f_\theta(x_i)) - \log|\det J| \tag{5}$$

This expression can be optimized using standard backpropagation and stochastic gradient descent.

This RVT approach requires an invertible function $f_\theta(\cdot)$ with a tractable Jacobian determinant. Autoregressive neural networks are a solution that arbitrarily orders the dimensions (e.g., pixels) and models each as a normal distribution conditioned on the preceding dimensions [15, 16]:

$$\begin{aligned} x_i &\sim \mathcal{N}(0,1) \\ y_i &= x_i \exp s(\boldsymbol{x}_{1:i-1}) + t(\boldsymbol{x}_{1:i-1}) \end{aligned} \tag{6}$$

where y_i is the ith ordered dimension of N-dimensional output $\boldsymbol{y}_{1:N}$, $s(\cdot)$ and $t(\cdot)$ are neural networks predicting the normal distribution parameters from the previous $i-1$ dimensions of $\boldsymbol{y}$. Equation (6) can be inverted easily:

$$x_i = [y_i - t(\boldsymbol{x}_{1:i-1})] \exp -s(\boldsymbol{x}_{1:i-1}) \tag{7}$$

The Jacobian of Eq. (6) is $J_{ij} = \partial y_i / \partial x_j$. Note that $J_{ij} = 0$ for $j > i$ because y_i only depends on x_i and $\boldsymbol{x}_{1:i-1}$. Therefore, J is lower triangular, and the tractable determinant is

$$|\det J| = \prod_i \frac{\partial y_i}{\partial x_i} = \exp \sum_i s(\boldsymbol{x}_{1:i-1}) \tag{8}$$

Stacking several autogressive networks end-to-end ($\boldsymbol{y}$ of the previous block is $\boldsymbol{x}$ of the next block) creates a powerful autoregressive flow [15]. While highly capable, this autogressive strategy unavoidably requires inefficient sequential computation in either the forward/normalizing [15] or inverse/sampling [16] direction.

Trading power for efficiency, affine coupling blocks [14, 17] use two partitions $\boldsymbol{x} = [\boldsymbol{x}_S, \boldsymbol{x}_{\bar{S}}]$ instead of autogressively calculating each x_i [16]:

$$\begin{aligned} \boldsymbol{y}_S &= \boldsymbol{x}_S \\ \boldsymbol{y}_{\bar{S}} &= \boldsymbol{x}_{\bar{S}} \odot \exp s(\boldsymbol{x}_S) + t(\boldsymbol{x}_S) \end{aligned} \tag{9}$$

The block's output is $\boldsymbol{y} = [\boldsymbol{y}_S, \boldsymbol{y}_{\bar{S}}]$, and $\odot$ is the elementwise product. Following Eq. 8, the corresponding Jacobian determinant is

$$|\det J| = \exp \sum s(\boldsymbol{x}_S), \tag{10}$$

and the inverted affine coupling block is simply

$$\begin{aligned} \boldsymbol{x}_S &= \boldsymbol{y}_S \\ \boldsymbol{x}_{\bar{S}} &= [\boldsymbol{y}_{\bar{S}} - t(\boldsymbol{x}_S)] \odot \exp -s(\boldsymbol{x}_S) \end{aligned} \tag{11}$$

Figure 1 graphically illustrates the forward and inverse operations. Without inefficient autoregressive steps, coupling blocks support fast computation in both directions. Stacking multiple coupling blocks end-to-end with varying (e.g., complementary) partitions enables information from different parts of the input to "mix" sufficiently and create an effective normalizing mapping $f_\theta(\cdot)$ such that $f_\theta(\boldsymbol{x}) = \boldsymbol{\epsilon} \sim q(\boldsymbol{\epsilon})$ [14]. Affine coupling blocks can be extended to model the conditional distribution $p(\boldsymbol{x} \mid \boldsymbol{c})$ by including the contextual information $\boldsymbol{c}$ as an additional argument to the scale and translate neural networks (see dotted path in Fig. 1). The conditional flow is then written $f_\theta(\boldsymbol{x}; \boldsymbol{c})$.

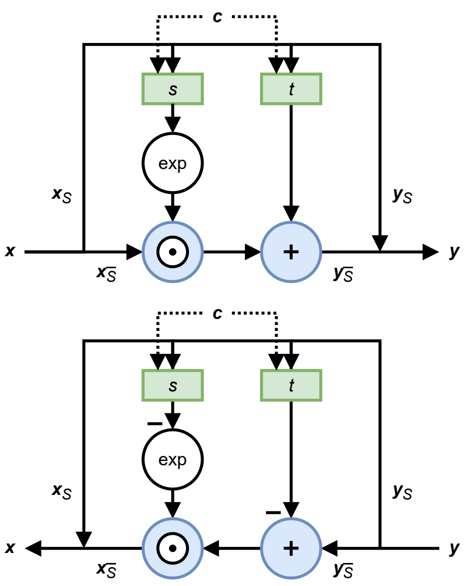


FIGURE 1: FORWARD (TOP) AND INVERSE (BOTTOM) AFFINE COUPLING BLOCK OPERATIONS

3. PROPOSED METHODS FOR MANUFACTURING

Normalizing flows are an attractive, multipurpose generative model for intelligent manufacturing because they are easier to train than GANs and VAEs, require only a single pass to generate samples (vs. diffusion networks), and can produce density estimates via the invertible flow transformation. Two specific manufacturing applications are generating synthetic training data and estimating parameters of a degradation model.

3.1 Synthetic Training Data Generation

Real-world manufacturing data sets contain very few examples of faults and defects. Synthesizing additional examples can balance the data set. A classifier trained on balanced data can find a decision boundary that generalizes better than a classifier trained on limited fault examples. Existing manufacturing work has used GANs to generate synthetic vibration data [18, 19] and surface defect images [20], but normalizing flows can accomplish the same tasks with simpler, more stable training objectives.

While applying flows to raw images requires complex hierarchical architectures to capture multiscale structures like edges, shapes, and objects [17, 21], Fig. 2 shows how a normalizing flow can instead model the latent code distribution of an image autoencoder, following the idea of Latent Diffusion Models (LDM) [6]. In this approach, a latent code is sampled from the normalizing flow and then decoded into the final output image. This leverages the strengths of both models—the autoencoder generates high-quality reconstructions while the flow models latent codes lacking high-dimensional complexities. For grayscale surface defect images, an autoencoder with encoder $h_\psi(\cdot)$ and decoder $g_\phi(\cdot)$ is pretrained using Binary Cross-Entropy (BCE) loss:

$$\mathcal{L}_{\mathrm{AE}}(\boldsymbol{x}_i) = \boldsymbol{x}_i \log \hat{\boldsymbol{x}}_i + (1 - \boldsymbol{x}_i) \log(1 - \hat{\boldsymbol{x}}_i) \tag{12}$$

where $\boldsymbol{x}_i$ is the ith input example, and $\hat{\boldsymbol{x}}_i = g_\phi(h_\psi(\boldsymbol{x}_i))$ is the reconstruction. The autoencoder is then frozen, and the normalizing flow $f_\theta(\cdot)$ is trained using loss derived from Eq. (5):

$$\mathcal{L}_{\mathrm{NF}}(\boldsymbol{x}_i) = -\log q(f_\theta(h_\psi(\boldsymbol{x}_i))) - \log|\det J| \tag{13}$$

Since $q(\epsilon)$ is frequently chosen to be $\mathcal{N}(\boldsymbol{0}, I)$, $\log q(\boldsymbol{\epsilon}) = -1/2 \cdot (\log 2\pi + \epsilon^2)$. After training, synthetic images can be generated by sampling the base distribution, inverting the flow, and applying the decoder:

$$\begin{aligned} \boldsymbol{\epsilon} &\sim \mathcal{N}(\boldsymbol{0}, I) \\ \boldsymbol{x} &= g_\phi(f_\theta^{-1}(\boldsymbol{\epsilon})) \end{aligned} \tag{14}$$

The synthetic images can balance the data set for training classifiers.

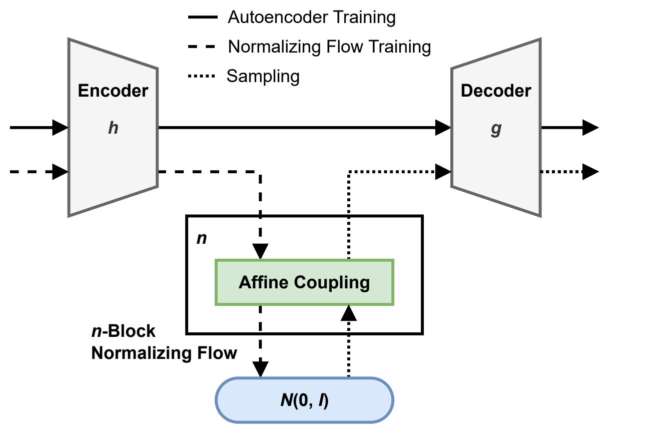


FIGURE 2: FLOW-BASED SURFACE DEFECT IMAGE GENERATION

3.2 Degradation Model Parameter Estimation

Estimating the RUL of manufacturing components is an important part of predictive maintenance. Degradation trends can often be modelled empirically (e.g., with an exponential curve) and the model parameters inferred from observations. For example, the parameters of an empirical model of milling machine tool wear can be estimated from a series of wear observations. The estimated parameters can then be used for prognosis, i.e., prediction of tool wear evolution in future cycles. If $p(z \mid \boldsymbol{x})$ represents the distribution of model parameters z when data $\boldsymbol{x}$ is observed, this inference can be mathematically expressed as:

$$p(z \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid z)p(z)}{p(\boldsymbol{x})} \tag{15}$$

Here $p(\boldsymbol{x} \mid z)$ is the likelihood of observation $\boldsymbol{x}$ given a model with parameters z, $p(z)$ is the prior over the model parameters, and $p(\boldsymbol{x}) = \int p(\boldsymbol{x} \mid z)p(z)dz$ is the likelihood function of the data. In general, $p(\boldsymbol{x})$ is intractable because $p(\boldsymbol{x} \mid z)$ and $p(z)$ are not restricted to conjugate pairs. Bayesian methods for approximate inference like Markov chain Monte Carlo (MCMC) and Particle Filtering (PF) can estimate the posterior $p(z \mid \boldsymbol{x})$ but are computationally intense and perform inference from scratch on each new $\boldsymbol{x}$.

A more efficient approach is amortized inference, in which global parameters (e.g., the weights of a neural network) are learned and reused to quickly predict local parameters (e.g., the parameters of the latent variable distributions) from observations [22]. Normalizing flows can solve this problem by modeling $p(z \mid \boldsymbol{x})$ based on the distribution of Monte Carlo simulations $p(\boldsymbol{x} \mid z)$ with parameters sampled from the prior $p(z)$ [23]. These simulations form a data set of pairs $(z_i, \boldsymbol{x}_i)$ from $p(\boldsymbol{x} \mid z)p(z)$ (step 1 in Fig. 3). In practice, time-series observations may not have fixed length. Therefore, a summary network $h_\varphi(\cdot)$ condenses variable-length observations into fixed-length context vectors [23]. The conditional normalizing flow is trained with the following loss function (step 2 in Fig. 3):

$$\mathscr{L}_{\text{cNF}}(z_i, \boldsymbol{x}_i) = -\log q(f_\theta(z_i; h_\varphi(\boldsymbol{x}_i))) - \log|\det J| \tag{16}$$

Approximate inference on new observation $\boldsymbol{x}$ is possible by sampling the learned posterior distribution of model parameters (step

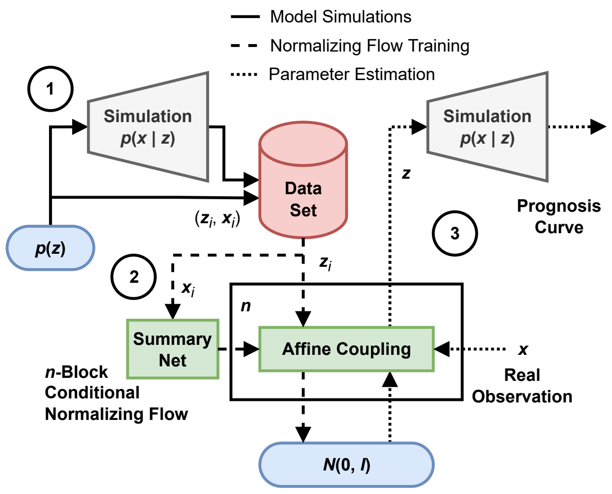


FIGURE 3: PARAMETER ESTIMATION FOR RUL PROGNOSIS

3 in Fig. 3):

$$\begin{aligned} \boldsymbol{\epsilon} &\sim \mathcal{N}(\boldsymbol{0}, I) \\ z &= f_\theta^{-1}(\boldsymbol{\epsilon}; h_\varphi(\boldsymbol{x})) \end{aligned} \tag{17}$$

The stochastic model $p(\boldsymbol{x} \mid z)$ can then use the posterior samples to generate prognosis curves. Importantly, this amortized inference is well-suited for manufacturing applications since it does not require real observations to train the inference model.

4. SYNTHETIC DEFECT IMAGE EXPERIMENTS

To validate the proposed image generation methods from Section 3.1, experiments were conducted to generate synthetic product surface defect images to augment a highly imbalanced manufacturing data set.

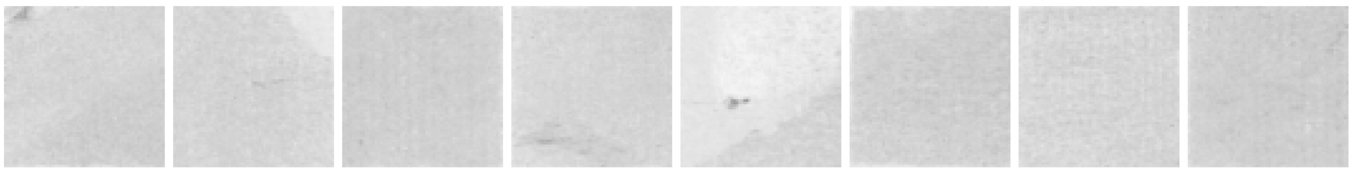
4.1 Kolektor SDD2 Data Set

The experiment uses the publicly available Kolektor SDD2 data set of product surface defect images from the Kolektor Group d.o.o. [24]. The training set contains 246 images with defects and 2085 images without defects, approximately a 10:1 imbalance. Synthetic defect images would help balance this data set. To learn the defect image distribution, the 246 defect images are used as the training set. Images are resized to 64×64, converted to grayscale, randomly flipped horizontally, and uniformly dequantized into the range [0, 1]. Figure 4a shows examples of real defect images used for training.

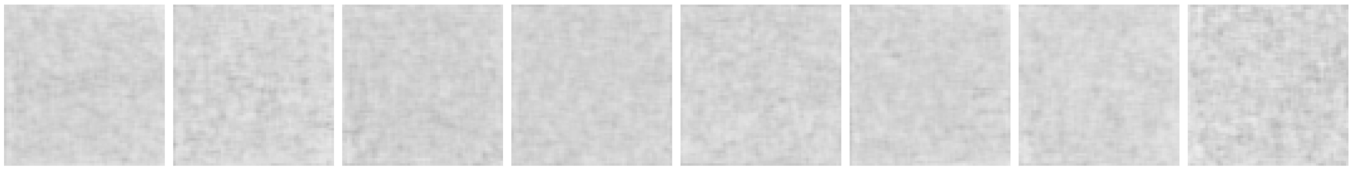
4.2 Network Design and Training

The network architecture uses Fig. 2 as a template. Figure 5 shows the details of the autoencoder, and Fig. 6 shows the the details of the normalizing flow that operates on the 16-length latent codes of the autoencoder and uses alternating checkboard masks for the partitions. Note that the autoencoder uses tanh activation for the latent code, producing constrained values easier for the flow to model. The scale output in the flow blocks uses a modified tanh activation that includes a trainable parameter α that mitigates numerical instability with the exponential function [25]:

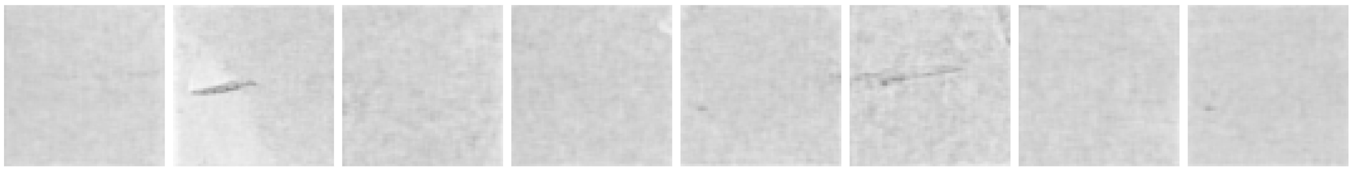
$$\tanh_\alpha(x) = \alpha \tanh(x/\alpha) \tag{18}$$



(a) Surface defect training images



(b) Synthetic images from AE latent space



(c) Synthetic images from normalizing flow

FIGURE 4: EXAMPLE IMAGES FROM SURFACE DEFECT EXPERIMENTS

The autoencoder is pre-trained for 300 epochs using a learning rate of 0.0001, BCE loss from Eq. (12), and the Adam optimizer. The encoder and decoder are then frozen, and the flow is trained on latent codes for 4700 epochs using the loss function from Eq. (13). The learning rate starts at 0.001 and was halved every 500 epochs. Once trained, the network produces synthetic images by sampling a standard normal distribution, inverting the normalizing flow, and then decoding the latent code into an image (see Eq. (14)).

4.3 Synthetic Image Results

To evaluate the usefulness of the normalizing flow, the synthetic images are compared to images generated by decoding normally distributed latent codes without the normalizing flow (see Fig. 4b):

$$\begin{aligned} \boldsymbol{\epsilon} &\sim \mathcal{N}(\mathbf{0}, I) \\ \boldsymbol{x}' &= g_{\phi}(\boldsymbol{\epsilon}) \end{aligned} \tag{19}$$

Compared to the real images from Fig. 4a, these examples exhibit fewer large-scale defects and more local, cloud-like textures. In contrast, the normalizing flow's synthetic images in Fig. 4c display local uniformness and larger defect features that are characteristic of the images from the training set. Quantative assessment of image quality lacks a commonly accepted metric and is left to future work.

These results motivate two observations. First, the latent codes of the AE are not normally distributed. If they were, the images from normally distributed codes would closely resemble real images, whereas the actual decoded images show both global and local differences. Conversely, the realistic images from the normalizing flow indicate that it can reasonably approximate the true distribution of latent codes. Thus, the normalizing flow facilitates efficient generation of synthetic surface defect images without needing complex dual-optimizer training like GAN or competing objectives like VAE.

5. TOOL WEAR PROGNOSIS EXPERIMENTS

To validate the proposed parameter estimation methods from Section 3.2, experiments were conducted using a conditional normalizing flow to estimate the distribution of degradation model parameters from a sequence of tool wear measurements. Future degradation can then be predicted using the estimated parameters.

5.1 Milling Data Set

The reference data set for milling tool wear degradation is distributed by the Prognostics Center of Excellence at NASA Ames [26]. The data set was created by UC Berkeley Emergent Space Tensegrities (BEST) Lab and includes cutting experiments on stainless steel and cast iron with depths of 0.75 mm and 1.5 mm and feed rates of 0.25 mm/s and 0.5 mm/s. Tool flank wear was measured throughout repeated cutting runs, although not always consistently. Missing data was linearly interpolated from neighboring values. Figure 7 shows a collection of flank wear degradation curves from the data set.

Given the exponential degradation trends, an exponential stepwise model is designed to capture tool wear evolution:

$$\begin{cases} x[0] &= 0 \\ x[k+1] &= 0.4b\exp(0.4ak) + x[k] + \nu \end{cases} \tag{20}$$

The coefficients of 0.4 are empirically selected to bias the model towards the range of the milling curves in Fig. 7, and ν is process/measurement noise drawn from $\mathcal{N}(0, 0.002)$. Probabilistic parameter estimation seeks to infer the posterior distribution of a and b conditioned on an observed sequence of wear measurements (e.g., the first 10 cuts). Values of a and b from the inferred

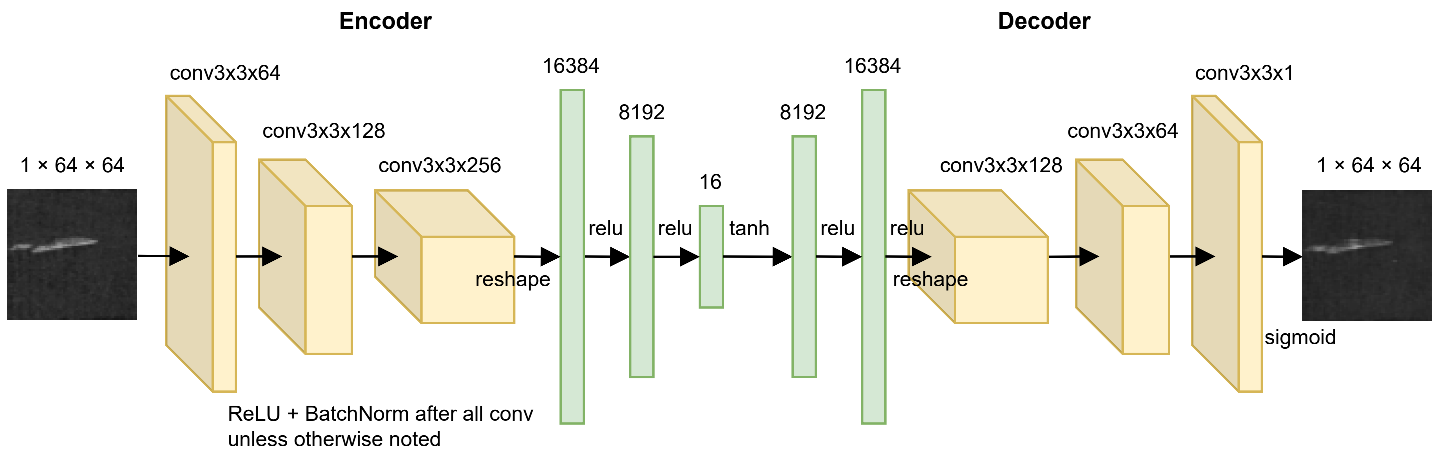


FIGURE 5: SYNTHETIC DEFECT IMAGE AUTOENCODER

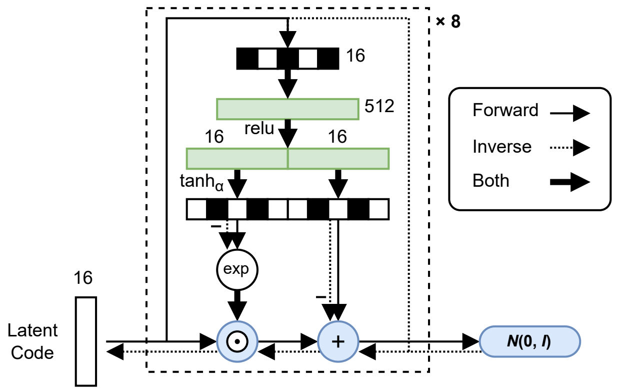


FIGURE 6: SYNTHETIC IMAGE NORMALIZING FLOW

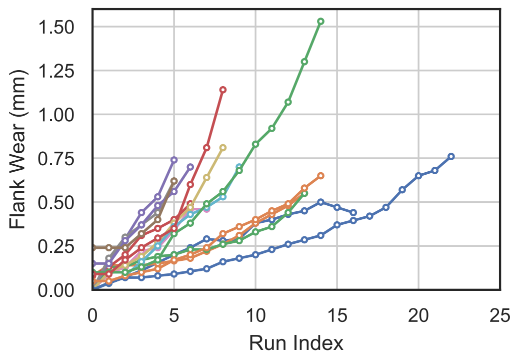


FIGURE 7: MILLING TOOL DEGRADATION CURVES

posterior can be substituted into Eq. (20) for Monte Carlo simulation of future degradation trends.

5.2 Simulated Degradation Data Set

To perform amortized inference with conditional normalizing flows, the normalizing flow is first trained on a simulated data set of the stepwise model Eq. (20) that captures the prior beliefs about the distribution of the two parameters a and b. Since the two parameters can exhibit joint behavior and dependence, the simulated data set was created by first generating curves for $k = 0, 1, \ldots, 24$ with a and b sampled from $\mathcal{N}(0, 1)$ and then excluding physically unexpected curves (i.e., curves with wear increasing too sharply, decreasing wear, or non-exponential wear trends). This formed a data set of simulations capturing these emprical prior beliefs about milling tool wear evolution.

5.3 Network Design and Training

Figure 8 shows the architecture of the conditional normalizing flow for modeling the posterior distribution of milling tool wear model parameters. A Gated Recurrent Unit (GRU) network is used to summarize variable-length tool wear observations into fixed-length context vectors for the conditional normalizing flow. A set of 243 hyperparameter grid search experiments were conducted with 2000 training and 200 validation examples and downselected to the top sixteen performers (see Table 1). The final hyperparameter set was selected as the trial with the lowest variation in loss (i.e., most stable loss curve). The resulting architecture is trained with 12000 training and 3000 validation simulated degradation curves. During training the summary network generates context vectors from the first 8 to 20 points of the simulated curve to support prognosis with variable-length observations. The entire model is trained five separate times with differing random seeds to assess variability.

After training, prognosis curves are generated by mapping an observed milling degradation curve into a context vector with the GRU summary network and conditioning the normalizing flow samples on this context. Repeatedly sampling the conditional normalizing flow with fixed context produces a group of possible parameter sets (a, b) from $p(a, b \mid \boldsymbol{x})$ which can be used with Eq. (20) to predict future evolution. Prognosis of future tool wear degradation is performed after 10 cuts in these experiments.

5.4 Parameter Estimation Results

Figure 9 shows the prognosis performance after 10 cuts on the randomly selected validation set of milling tool wear curves (milling cases 1, 3, and 13). While two of the actual curves end up

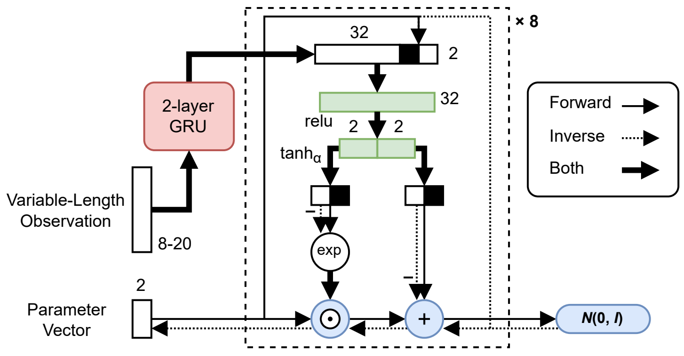


FIGURE 8: TOOL WEAR PARAMETER ESTIMATION

TABLE 1: TOOL WEAR HYPERPARAMETER SEARCH

Hyperparameter	Search Values	Selected
Learning Rate	[1e-4, 1e-3, 5e-3]	1e-3
GRU Layers	[2, 3, 4]	2
GRU Output Size	[8, 16, 32]	32
Number of Affine Coupling Blocks	[8, 16, 32]	8
Affine Coupling Block Hidden Layer Size	[8, 16, 32]	32

outside the 90%/10% quantiles of prognosis, this behavior stems from the strictly exponential nature of the stepwise model. The model assumes that degradation will have increasing slope, but the problematic curves have sections where degradation levels off or even decreases. This could be caused by measurement error which future work can capture by adjusting the stepwise model. Root Mean Square Error (RMSE) is computed for each milling example using the median of 10,000 prognosis curves. The uncertainty is captured by repeating this for the five models trained with different random seeds. In all cases, the RMSE falls well below 0.1 mm.

Figure 10 shows the prognosis performance after 10 cuts on the randomly selected test set (milling cases 2, 11, and 12). The output range grows as the exponential curves are propagated farther into the future, causing longer curves to have wider windows of uncertainty. However, the median of the prognosis curves closely and consistently matches the actual tool wear evolution, as evidenced by the low RMSE. All RMSE values fall below 0.05 mm with standard deviations less than 0.01 mm. This indicates that the GRU summary network has learned a meaningful context representation, and the flow can use this context to approximate the posterior distribution of model parameters. While the parameters could be estimated with MCMC or PF with comparable or better RMSE, inference with normalizing flows is more efficient—the same flow can estimate parameters from all new curves without retraining. In contrast, PF requires sequential updates of hundreds or thousands of particles for every curve, and MCMC requires thousands of model simulations, many of which must be rejected to uphold the sampling criteria. Thus, the

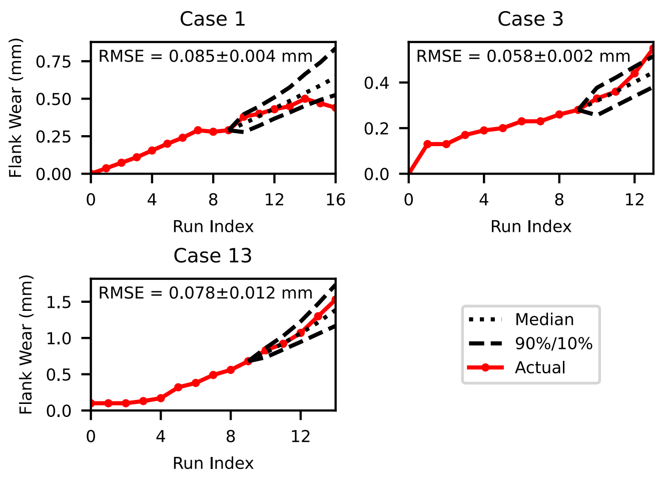


FIGURE 9: VALIDATION WEAR PROGNOSIS AFTER 10 CUTS

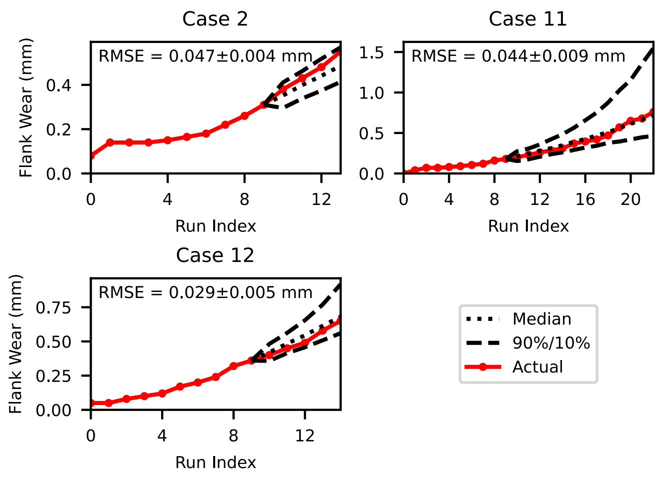


FIGURE 10: TEST WEAR PROGNOSIS AFTER 10 CUTS

normalizing flow-based approach offers a reusable solution that overcomes the computational burdens of both traditional techniques.

6. CONCLUSION

Experiments show that normalizing flows can generate realistic synthetic surface defect images capturing global and local image structures. These images can augment an imbalanced data set, future work in this area should evaluate the quantitative impact of using the synthetic images when training a defect classfier. Similarly, parameter estimation experiments show that normalizing flows are effective for amortized inference with milling tool wear degradation models. The posterior parameter distribution generated by the conditional normalizing flow produces prognosis curves accurately capturing future trends. To further this area, future studies should explore using flows for more complex multistage/piecewise degradation models. Both case studies demonstrate that normalizing flows can be a general approach for solving diverse manufacturing and condition monitoring problems. The advantages include more stable training compared to

GANs and VAEs, efficient generation versus diffusion models, and fast, amortized inference instead of time-consuming iterations of MCMC or PF. Additional research should continue exploring other manufacturing applications of normalizing flows such as likelihood-based anomaly detection.

ACKNOWLEDGMENTS

This work is supported by the National Science Foundation under Grant No. 2015889. We would thank the University of Kentucky Center for Computational Sciences and Information Technology Services Research Computing for their support and use of the Lipscomb Compute Cluster and associated research computing resources.

REFERENCES

[1] Goodfellow, Ian J., Pouget-Abadie, Jean, Mirza, Medhi, Xu, Bing, Warde-Farley, David, Ozair, Sherjil, Courville, Aaron and Bengio, Yoshua. "Generative Adversarial Nets." *ArXiv e-prints* (2016). URL https://arxiv.org/abs/1406.2661.

[2] Kingma, Diederik P and Welling, Max. "Auto-Encoding Variational Bayes." *ArXiv e-prints* (2013). URL https://arxiv.org/abs/1312.6114.

[3] Rezende, Danilo Jimenez, Mohamed, Shakir and Wierstra, Dann. "Stochastic Backpropagation and Approximate Inference in Deep Generative Models." *Proceedings of the 31st International Conference on Machine Learning*. Beijing, China, June 21–26, 2014.

[4] Sohl-Dickstein, Jascha, Weiss, Eric A., Maheswaranathan, Niru and Ganguli, Surya. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics." *Proceedings of the 32nd International Conference on Machine Learning*. Lille, France, July 6–11, 2015.

[5] Ramesh, Aditya, Dhariwal, Prafulla, Nichol, Alex, Chu, Casey and Chen, Mark. "Hierarchical Text-Conditional Image Generation with CLIP Latents." (2022)URL https://arxiv.org/abs/2204.06125.

[6] Rombach, Robin, Blattmann, Andreas, Lorenz, Dominik, Esser, Patrick and Ommer, Björn. "High-Resolution Image Synthesis with Latent Diffusion Models." *ArXiv e-prints* (2022). URL https://arxiv.org/abs/2112.10752.

[7] Zhang, Liangwei, Lin, Jing, Shao, Haidong, Zhang, Zhicong, Yan, Xiaohui and Long, Jianyu. "End-to-end unsupervised fault detection using a flow-based model." *Reliability Engineering and System Safety* Vol. 215 (2021): p. 107805.

[8] Rudolph, Marco, Wandt, Bastian and Rosenhahn, Bodo. "Same Same But DifferNet: Semi-Supervised Defection Detection with Normalizing Flows." *IEEE Winter Conference on Applications of Computer Vision (WACV)*: pp. 1906–1915. Virtual Event, January 3–8, 2021.

[9] Szarski, Martin and Chauhan, Sunita. "An unsupervised defect detection model for a dry carbon fiber textile." *Journal of Intelligent Manufacturing* Vol. 33 (2022): pp. 2075–2092.

[10] Yang, Haosen, Ding, Keqin, Qiu, Robert C. and Mi, Tiebin. "Remaining Useful Life Prediction Based on Normalizing Flow Embedded Sequence-to-Sequence Learning." *IEEE Transactions on Reliability* Vol. 70 No. 4 (2021): pp. 1342–1354.

[11] Salimans, Tim, Goodfellow, Ian, Zremba, Wojciech, Cheung, Vicki, Radford, Alec and Chen, Xi. "Improved Techniques for Training GANs." *Advances in Neural Information Processing 29 (NIPS 2016)*: pp. 2234–2242. Barcelona, Spain, December 5–10, 2016.

[12] Higgins, Irina, Matthey, Loic, Pal, Arka, Burgess, Christopher, Glorot, Xavier, Botvinick, Matthew, Mohamed, Shakir and Lerchner, Alexander. "β-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework." *5th International Conference on Learning Representations*. Toulon, France, April 24–26, 2017.

[13] van den Oord, Aaron, Vinyals, Oriol and Kavukcuoglu, Koray. "Neural Discrete Representation Learning." *Advances in Neural Information Processing Systems 30 (NIPS 2017)*: pp. 6306–6315. Long Beach, CA, December 4–9, 2017.

[14] Dinh, Laurent, Krueger, David and Bengio, Yoshua. "NICE: Non-linear Independent Components Estimation." *3rd International Conference on Learning Representations*. San Diego, CA, May 7–9, 2015.

[15] Kingma, Diederik P., Salimans, Tim, Jozefowicz, Rafal, Chen, Xi, Sutskever, Ilya and Welling, Max. "Improved Variational Inference with Inverse Autoregressive Flow." *Advances in Neural Information Processing 29 (NIPS 2016)*: pp. 4743–4751. Barcelona, Spain, December 5–10, 2016.

[16] Papamakarios, George, Pavlakou, Theo and Murray, Iain. "Masked Autoregressive Flows for Density Estimation." *Advances in Neural Information Processing Systems 30 (NIPS 2017)*: pp. 2338–2347. Long Beach, CA, December 4–9, 2017.

[17] Dinh, Laurent, Sohl-Dickstein, Jascha and Bengio, Samy. "Density estimation using Real NVP." *5th International Conference on Learning Representations*. Toulon, France, April 24–26, 2017.

[18] Luo, Jia, Huang, Jinying and Li, Hongmei. "A case study of conditional deep convolutional generative adversarial networks in machine fault diagnosis." *Journal of Intelligent Manufacturing* Vol. 32 (2021): pp. 407–425.

[19] Ma, Liang, Ding, Yu, Wang, Zili, Wang, Chao, Ma, Jian and Lu, Chen. "An interpretable data augmentation scheme for machine fault diagnosis based on a sparsity-constrained generative adversarial network." *Expert Systems with Applications* Vol. 182 (2021): p. 115234.

[20] Niu, Shuanlong, Li, Bin, Wang, Xinggang and Lin, Hui. "Defect Image Sample Generation With GAN for Improving Defect Recognition." *IEEE Transactions on Automation Science and Engineering* Vol. 17 (2020): pp. 1611–1622.

[21] Kingma, Diederik P. and Dhariwal, Prafulla. "Glow: Generative Flow with Invertible 1x1 Convolutions." *Advances in Neural Information Processing Systems 31 (NeurIPS 2018)*: pp. 10215–10224. Montreal, CA, December 2–8, 2018.

[22] Blei, David M., Kucukelbir, Alp and McAuliffe, Jon D. "Variational Inference: A Review for Statisticians." *Journal*

of the American Statistical Association Vol. 112 (2017): pp. 859–877.

[23] Radev, Stefan T., Mertens, Ulf K., Voss, Andreas, Ardizzone, Lynton and Köthe, Ullrich. "BayesFlow: Learning Complex Stochastic Models With Invertible Neural Networks." *IEEE Transactions on Neural Networks and Learning Systems* Vol. 33 No. 4 (2022): pp. 1452–1466.

[24] Božič, Jakob, Tabernik, Domen and Skočaj, Danijel. "Mixed supervision for surface-defect detection: From wekly to full supervised learning." *Computers in Industry* Vol. 129 (2021): p. 103459.

[25] Lippe, Phillip. "Tutorial 11: Normalizing Flows for image modeling." UvA Deep Learning Tutorials (2022). Accessed November 10, 2022, URL https://uvadlc-notebooks.readthedocs.io/en/latest/tutorial_notebooks/tutorial11/NF_image_modeling.html.

[26] Agogino, A. and Goebel, K. "Milling Data Set." (2007). BEST Lab, UC Berkeley. NASA Prognostics Data Repository, NASA Ames Research Center, Moffett Field, CA.