machines

MDPI

Article

Effects of Window and Batch Size on Autoencoder-LSTM Models for Remaining Useful Life Prediction

Eugene Jeon [1], Donghwan Jin [2] and Yeonhee Kim [1,*]

[1] Department of Semiconductor Engineering, Hoseo University, Asan 31499, Republic of Korea; 20255240@vision.hoseo.edu
[2] MyMeta Co., Ltd., Seoul 08790, Republic of Korea; daniel@e-mymeta.com
* Correspondence: kimyh@hoseo.edu

Abstract

Remaining useful life (RUL) prediction is central to predictive maintenance, but acquiring sufficient run-to-failure data remains challenging. To better exploit limited labeled data, this study investigates a pipeline combining an unsupervised autoencoder (AE) and supervised LSTM regression on the NASA C-MAPSS dataset. Building on an AE-LSTM baseline, we analyze how window size and batch size affect accuracy and training efficiency. Using the FD001 and FD004 subsets with training-capped RUL labels, we perform multi-seed experiments over a wide grid of window lengths and batch sizes. The AE is pre-trained on normalized sensor streams and reused as a feature extractor, while the LSTM head is trained with early stopping. Performance was assessed using RMSE, C-MAPSS score, and training time, reporting 95% confidence intervals. Results show that fine-tuning the encoder with a batch size of 128 yielded the best mean RMSE of 13.99 (FD001) and 28.67 (FD004). We obtained stable optimal window ranges (40–70 for FD001; 60–80 for FD004) and found that batch sizes of 64–256 offer the best accuracy–efficiency trade-off. These optimal ranges were further validated using Particle Swarm Optimization (PSO). These findings offer practical recommendations for tuning AE-LSTM-based RUL prediction models and demonstrate that performance remains stable within specific hyperparameter ranges.

Keywords: remaining useful life; unsupervised learning; hyperparameter optimization; statistical validation; stability analysis

1. Introduction

As manufacturing systems grow in complexity, predictive maintenance (PdM) for high-reliability industrial assets such as semiconductor manufacturing equipment and high-precision machinery has emerged as a critical strategy for maintaining operational stability and cost efficiency [1]. Unexpected equipment failures can lead to production delay, degraded product quality, and financial losses [2]. Parallel challenges exist for other high-reliability assets, including turbofan engines used in aerospace applications, where unplanned downtime is similarly costly and presents safety-critical risks [3]. To mitigate such risks, recent studies have increasingly focused on data-driven models that estimate the remaining useful life (RUL) of critical components and inform timely maintenance decisions [4].

Deep learning-based approaches now represent a mainstream strategy in RUL prediction and often achieve high accuracy when sufficient run-to-failure data are available for supervised learning [4,5]. In real-world industrial environments, however, failures are

Check for updates

Academic Editors: Diwang Ruan, Jianping Yan, Mengdi Li and Junxing Li

Received: 28 December 2025
Revised: 20 January 2026
Accepted: 22 January 2026
Published: 23 January 2026

Copyright: © 2026 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license.

typically rare events, and collecting degradation trajectories that cover the entire lifetime of equipment is both costly and time-intensive. Consequently, labeled run-to-failure datasets are often scarce, whereas large volumes of unlabeled condition monitoring data are continuously accumulated during normal operation [5]. This data imbalance has spurred investigations into unsupervised and semi-supervised methods that can learn informative latent representations from unlabeled sensor streams and then adapt them for RUL prediction with limited labels [5,6].

Autoencoders and sequence models such as long short-term memory (LSTM) have been frequently employed in this context [7]. An autoencoder can compress multivariate sensor signals into low-dimensional latent health indicators, while an LSTM can capture temporal degradation patterns from these representations [5,8]. More recently, advanced architectures such as attention mechanisms and Transformers have been proposed to enhance prediction accuracy [4]. However, greater architectural complexity does not guarantee better performance in real applications, especially when training data are limited (e.g., in specialized, low-volume manufacturing) and computational resources are constrained [8,9]. In many cases, the choice of learning settings, particularly key hyperparameters, is often as critical as the model architecture itself.

Among various hyperparameters, the sequence window size and mini-batch size are of particular importance in RUL prediction [10,11]. The window size determines the temporal context provided to the model, influencing its ability to capture early degradation signatures and long-term dependencies [4,8]. The batch size directly affects the stochastic optimization dynamics, generalization behavior, and training efficiency [12]. Nevertheless, in many existing RUL studies, these hyperparameters are typically treated as simple tuning knobs, and only a single selected configuration is reported [13]. For instance, many deep learning-based approaches simply adopt fixed window sizes (e.g., 30 or 50 cycles) based on empirical conventions without exploring the sensitivity of the model to this parameter. Although meta-heuristic algorithms such as Particle Swarm Optimization (PSO) or Genetic Algorithms (GA) are capable of locating optimal values, they often function as "black boxes" that obscure the underlying performance landscape, failing to reveal the sensitivity and stability of the parameters [14,15]. Systematic analyses of how different combinations of window size and batch size jointly shape the trade-off between prediction performance and computational cost remain limited, especially in hybrid models that combine unsupervised representation learning with supervised regression [5,13].

To address this gap, this study empirically investigates the impact of window size and batch size on RUL prediction using a simple autoencoder-LSTM (AE-LSTM) model that utilizes unlabeled sensor data for feature extraction and labeled run-to-failure trajectories for regression [11,16]. We construct an unsupervised-supervised pipeline in which an autoencoder is first trained on normalized sensor sequences from the NASA C-MAPSS turbofan engine dataset and then reused as a feature extractor within an LSTM-based regression model [5,17,18]. The LSTM head is trained to predict RUL targets using the standard capped-label protocol with training RUL values limited to 125 cycles [8]. We conduct extensive multi-seed experiments on the FD001 and FD004 subsets, systematically varying both window size and batch size over a wide grid. We then quantitatively evaluate their effects on predictive accuracy and training efficiency using root mean squared error, the asymmetric C-MAPSS scoring function, and per-epoch training time. Statistical validation is performed across multiple random seeds to ensure robustness [4,8].

The main contributions of this study are threefold. First, we present a straightforward but effective AE-LSTM pipeline that explicitly separates unsupervised representation learning from supervised RUL regression, allowing unlabeled sensor streams to be utilized alongside limited failure labels [5]. Rather than proposing a new architecture, we focus

on how this commonly used model behaves under different temporal and optimization hyperparameters. Second, our extensive experimental study on the C-MAPSS FD001 and FD004 subsets characterizes how window size and batch size jointly influence both predictive accuracy and the per-epoch training time of the supervised AE-LSTM stage, revealing dataset-dependent yet consistent optimal ranges for these hyperparameters [13,19]. Third, from these results, we derive empirical guidelines and accuracy–efficiency trade-off insights. Unlike heuristic algorithms that identify a single optimal point, our comprehensive search maps the stability landscape, identifying distinct hyperparameter regions where the model performance remains stable and robust [14]. Furthermore, we cross-validate these empirical findings using Particle Swarm Optimization (PSO) to confirm that the identified stable regions encompass the global optima found by the algorithm [15]. These insights—where efficiency is measured in terms of supervised-stage epoch time—can assist practitioners in selecting hyperparameter configurations according to their application priorities, such as favoring higher accuracy or reduced training cost.

2. Theoretical Background and Related Work

2.1. Deep Learning-Based RUL Prediction

Deep learning-based approaches have become a dominant paradigm for data-driven RUL prediction. Early work by Mitici et al. [20] showed that LSTM networks can effectively model long-term temporal dependencies in the C-MAPSS turbofan dataset and outperform traditional shallow models for RUL estimation. Subsequent studies extended this line of research by exploring GRU architecture and hybrid CNN-RNN models to better capture local degradation patterns and multiscale temporal dynamics in sensor streams [21]. More recent work introduced attention mechanisms and Transformer-based architectures, often reporting improved accuracy on C-MAPSS and related benchmarks at the cost of increased architectural complexity and computational demand [4,21].

In parallel, several studies have proposed deep LSTM variants tailored to specific industrial assets, such as power electronics and rotating machinery, sometimes combined with transfer learning or ensemble strategies to alleviate data scarcity [7]. Overall, this body of work demonstrates that RUL can be accurately predicted when sufficient labeled run-to-failure trajectories are available. However, these supervised models are still fundamentally constrained by the limited availability of labeled degradation data in real PdM deployments, motivating the use of unsupervised representation learning and hybrid learning schemes [22].

2.2. Unsupervised Representation Learning and AE-LSTM Hybrids

To better exploit unlabeled condition-monitoring data, unsupervised learning and hybrid unsupervised-supervised pipelines have attracted increasing attention. Belay et al. [6] reviewed unsupervised anomaly detection methods for multivariate time series and highlighted that autoencoder-based models are widely adopted in prognostics and health management (PHM) and Internet of Things (IoT) settings, as they can compress high-dimensional sensor signals into compact latent representations and detect deviations via reconstruction error. In the RUL context, deep autoencoders have been used to construct latent health indicators that summarize degradation trends before applying a supervised regressor on top [5].

Several hybrid architectures train feature extractors and RUL predictors jointly. For example, İnce and Genc proposed a joint autoencoder-regressor network in which a CNN-based autoencoder and an LSTM regressor are trained end-to-end to estimate RUL, demonstrating the benefit of combining unsupervised feature learning with sequence modeling [4,6,8]. Other works, such as adversarial or variational AE-LSTM models,

similarly aim to learn degradation-aware latent spaces while directly supervising the RUL head [16,22].

In contrast to these fully end-to-end models, the present study adopts a simple AE-LSTM pipeline in which an autoencoder is first trained on unlabeled sensor windows and its encoder is subsequently reused within an LSTM-based regression model for RUL prediction [5,8]. We consider both freezing the encoder and fine-tuning it jointly with the LSTM head during supervised training, allowing us to examine how temporal hyperparameters such as window size and batch size affect accuracy and training efficiency under different encoder-update strategies [7]. This decoupled design separates unsupervised representation learning from supervised RUL regression, which is the main focus of our analysis.

2.3. Hyperparameter Configuration and Accuracy–Efficiency Trade-Offs

Beyond architectural choices, recent surveys on deep learning for RUL prediction and time-series modeling have emphasized that hyperparameters such as sequence window size, batch size, and learning rate can have an impact on performance comparable to that of the network structure itself [13]. Window size controls the temporal context presented to the model, influencing its ability to capture early degradation signatures and long-term trends, while batch size affects stochastic optimization dynamics, generalization behavior, and per-epoch training cost [8,23].

From a more general deep learning perspective, Keskar et al. [12] showed that very large batch sizes tend to drive optimization towards sharp minima that generalize poorly, whereas small-batch training often converges to flatter minima that exhibit better generalization, highlighting an intrinsic accuracy–efficiency trade-off in batch-size selection. However, only limited work has examined how such theoretical insights manifest in RUL prediction settings, especially under practical constraints such as capped RUL targets and multi-condition degradation patterns.

In the RUL literature, many studies implicitly tune temporal hyperparameters while primarily emphasizing novel architecture. For instance, Shen et al. [3] proposed an aero-engine RUL prediction model that combines an improved grey wolf optimizer with a 1D CNN, illustrating the trend toward complex hybrid models and meta-heuristic tuning strategies. Yet, even in such works, window length and batch size are typically reported only as final chosen values rather than being systematically analyzed. To the best of our knowledge, there is still a lack of comprehensive empirical studies that quantify how different combinations of window size and batch size jointly affect both predictive accuracy and training efficiency in AE-LSTM-type RUL models [11,22]. Addressing this gap is the primary objective of the present work. In summary, while hybrid architectures are promising, the interplay between temporal context (window size) and optimization dynamics (batch size) remains under-explored. This study aims to bridge this gap by systematically mapping the accuracy–efficiency landscape of AE-LSTM models.

3. Materials and Methods

3.1. Dataset

The Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) dataset from NASA was used as a benchmark for predicting the RUL of turbofan engines [19,20]. The dataset comprises multivariate time-series signals generated from virtual engine simulations under diverse operating conditions and fault modes, including 21 sensor measurements and 3 operational settings [19,20].

To evaluate robustness and generalization capability, two sub-datasets with distinct characteristics were selected:

- FD001: The simplest configuration with a single operating condition and a single fault mode (high-pressure compressor degradation). It provides 100 engine units for training and 100 for testing, and is treated as the baseline scenario [19,20].
- FD004: The most complex configuration, featuring six operating conditions and two fault modes (high-pressure compressor and fan degradation). It contains 249 engine units for training and 248 for testing. This subset is used to assess model robustness under more realistic operational variability [19]. The same grid of window sizes and batch sizes is applied to both FD001 and FD004 to enable a consistent comparison, without additional dataset-specific hyperparameter tuning. The detailed specifications of the FD001 and FD004 subsets are summarized in Table 1.

Table 1. Summary of the C-MAPSS datasets.

Dataset	Training Units	Test Units	Conditions	Fault Modes
FD001	100	100	1	HPC * Degradation
FD004	249	248	6	HPC * & Fan Degradation

* High-pressure compressor.

3.2. Data Preprocessing

A preprocessing workflow was designed to transform raw sensor signals into a suitable input format for the deep learning models.

- Feature selection: Among the 21 sensors, static sensors whose readings remained constant throughout all operational cycles (sensors 1, 5, 6, 10, 16, 18, and 19) were excluded, as they do not contribute to degradation estimation. In total, 17 features—14 dynamic sensors plus 3 operational settings—were retained as model inputs [4,8].
- RUL label generation and capping: For each engine instance in the training set, RUL was computed by subtracting the current cycle from its maximum operational cycle [4,10]. Following common practice in C-MAPSS studies, the maximum RUL was capped at 125 cycles, yielding a piecewise-linear target. The capped RUL labels were used for training; the official RUL labels provided with the test subsets were used for evaluation [4,10].
- Normalization: All features were normalized to the [0, 1] range using the MinMaxScaler implementation from scikit-learn (version 1.7.2, NumFOCUS, Austin, TX, USA) [24]. To avoid data leakage, the scaler was fitted exclusively on the training set and then applied to both the training and test sets [4–6].
- Sliding-window construction for training: Overlapping sequences of length W (stride = 1) were extracted from each time series in the training set [8,25]. Each sequence forms an input tensor $X \in \mathbb{R}^{W \times F}$, with target y set to the RUL at the last time step; here, W is the window size and $F = 17$ is the number of input features [4].
- Test window construction: For evaluation, a single window of length W was constructed per test engine by taking the last W time steps of the corresponding time series [19,20]. If the remaining sequence length was shorter than W, the beginning of the window was padded by repeating the earliest available measurements ("edge" padding). The model thus produces one RUL prediction per engine unit, which is compared against the corresponding ground-truth RUL label [19,22].

3.3. Model Architecture: Autoencoder-LSTM (AE-LSTM)

This study focuses on systematically analyzing the effects of key hyperparameters rather than proposing a novel architecture for RUL prediction. Accordingly, a hybrid AE-LSTM model, widely adopted as a baseline in this domain, was employed. The architecture integrates

an unsupervised feature extractor (AE) and a temporal predictor (LSTM), forming an effective testbed for evaluating the impact of temporal and optimization hyperparameters [5].

- Unsupervised feature extraction (AE): The AE takes 17-dimensional sensor vectors as input and compresses them into an 8-dimensional latent representation. The encoder consists of fully connected layers with the structure $17 \rightarrow 12 \rightarrow 8$, with ReLU activations after each linear layer. The decoder symmetrically reconstructs the input via layers $8 \rightarrow 12 \rightarrow 17$, again with ReLU between the hidden layers. The AE is trained in an unsupervised manner using mean squared reconstruction error. For each subset (FD001 and FD004), each window size, and each random seed, a separate AE is pre-trained on all sliding windows extracted from the training units [5]. The resulting encoder weights are then used to initialize the encoder inside the AE-LSTM head for all batch sizes and encoder training modes corresponding to that subset-window-seed configuration [5].
- RUL prediction (LSTM head): For the supervised stage, the pre-trained encoder is embedded into an AE-LSTM head. For each input window, the encoder is applied to every time step, producing a sequence of 8-dimensional latent vectors. These encoded sequences are fed into a two-layer LSTM with 50 hidden units per layer and a dropout rate of 0.2 between layers to capture temporal dependencies. The hidden state at the final time step is passed through a regression head comprising two fully connected layers $50 \rightarrow 25 \rightarrow 1$ with a ReLU activation between them to produce the RUL estimate.

Two encoder training strategies are considered during supervised learning. In the frozen-encoder mode, the AE encoder parameters are kept fixed after unsupervised pre-training, and only the LSTM layers and regression head are updated [10]. In the fine-tuned mode, the encoder is updated jointly with the LSTM head, but with a smaller learning rate (one quarter of the rate used for the LSTM and regression head parameters) to preserve the structure of the learned latent space [10].

For each subset and random seed, the AE was pre-trained exclusively on sliding windows extracted from the units used for supervised training, excluding the units reserved for validation. This "train-only AE pretraining" ensures that neither labels nor validation inputs are seen during the unsupervised pre-training stage [9].

The sizes of the latent space (8 units) and the LSTM hidden state (50 units with two layers) were chosen to follow commonly used AE-LSTM baselines for C-MAPSS RUL prediction while keeping the overall parameter count moderate [8], ensuring sufficient model capacity to capture degradation trends while maintaining a moderate parameter count to prevent overfitting [5]. Fixing these architectural hyperparameters allows the present study to isolate the effects of window size and batch size without conflating them with changes in model capacity. Under this architecture, the RUL predictor contains 33,701 trainable parameters when the encoder is included. The overall architecture is shown in Figure 1.

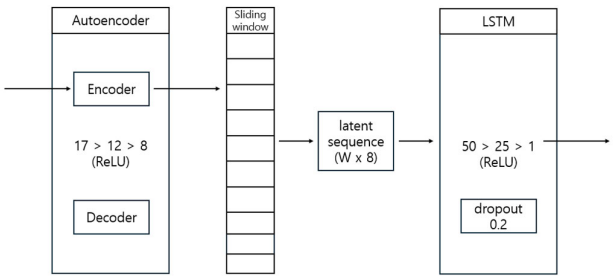


Figure 1. AE-LSTM structure diagram.

3.4. Experimental Setup

A series of experiments was conducted to examine the effects of window size, batch size, and encoder training strategy on both predictive accuracy and training efficiency. Instead of relying on a single run, each window-batch-encoder configuration was repeated over multiple random seeds to enable statistical analysis of performance variability.

Training for the AE and the AE-LSTM head was performed separately:

- AE pre-training: The AE was trained using the Adam optimizer with mean squared error loss and an initial learning rate of 0.001. For each subset (FD001 and FD004), each window size, and each random seed, a separate AE was pre-trained for 50 epochs on all sliding windows extracted from the training units used for supervised learning (excluding validation units). The resulting encoder weights were reused to initialize the encoder of the AE-LSTM head for all batch sizes and encoder training modes under the same subset-window-seed configuration [5,8].
- LSTM head training with early stopping: The AE-LSTM head was trained using mean squared error loss and the Adam optimizer. The maximum number of training epochs for the LSTM head was set to 150, but early stopping based on validation loss was applied with a patience of 15 epochs [22,23,25]. The learning-rate schedule included a warm-up phase during the first five epochs, linearly increasing the effective learning rate, and the base learning rate (0.001) was scaled linearly with the batch size relative to a reference batch size of 128 (following the Linear Scaling Rule), with an upper bound of 0.001 to prevent overly large steps [22,26]. This constraint ensures training stability but implies that for very large batch sizes (e.g., 512), the learning rate is effectively capped, which may limit the optimizer's ability to escape sharp minima compared to smaller batches. Gradient clipping with a threshold of 1.0 was applied to stabilize training. When fine-tuning the encoder, its learning rate was set to one quarter of the learning rate used for the LSTM and regression head parameters [27].
- Validation strategy and multi-seed experiments: For each subset, 20% of the training units were reserved as a validation set, ensuring that the validation set contained at least 1024 sliding windows for stable early stopping. For every configuration of window size, batch size, and encoder mode, the model was trained and evaluated across five random seeds [28]. For each seed, the same validation split was used to enable fair comparisons [9]. This multi-seed approach allows us to quantify and mitigate the bias arising from random weight initialization and optimization stochasticity [9]. Consequently, this ensures that the reported trends reflect the true impact of the hyperparameters rather than artifacts of a specific random state [12].

The main investigated and fixed hyperparameters are summarized in Table 2. Window size W was varied from 10 to 100 cycles in steps of 2, and batch size was varied over $\{32, 64, 128, 256, 512\}$, while all other hyperparameters were kept fixed to isolate the effects of these two parameters. The overall experimental workflow, encompassing data preprocessing and model training, is illustrated in Figure 2. All experiments were executed in a controlled computational environment, summarized in Table 3.

The lower and upper bounds of the window size grid (10 and 100 cycles) were selected to span short contexts that may miss part of the degradation trajectory and longer contexts that approach, but do not exceed, the commonly used RUL cap of 125 cycles. This range therefore emphasizes the more dynamic portion of the degradation phase while keeping the input length manageable [8,29]. The batch-size set {32, 64, 128, 256, 512} spans small to relatively large mini-batches that are practical for modern GPUs and representative of configurations commonly reported in C-MAPSS-based RUL studies [8,12,13]. Keeping all other architectural and optimization hyperparameters fixed makes it possible to attribute observed performance differences primarily to window size and batch size.

Table 2. Experimental Hyperparameter Settings.

Parameter	Values
Window Size	10–100 (step = 2)
Batch Size	32, 64, 128, 256, 512
Optimizer	Adam
Learning Rate	0.001 (base value)
AE Epochs	50
Max LSTM Epochs	150
Early stopping patience	15 epochs

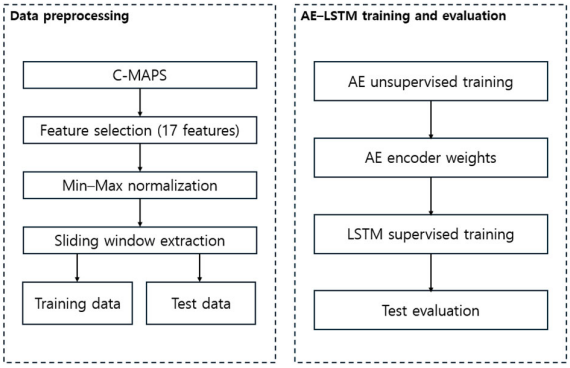


Figure 2. Preprocessing and training pipeline.

Table 3. Computational Environment.

Component	Specification
OS	Linux 5.15.153.1-microsoft-standard-WSL2
Docker version	4.45.0 (pytorch:2.2.1) (Docker Inc., Palo Alto, CA, USA)
Python Version	3.10.13 (Python Software Foundation, Beaverton, OR, USA)
PyTorch Version	2.2.1 (Linux Foundation, San Francisco, CA, USA)
CUDA Version	12.1 (NVIDIA Corp., Santa Clara, CA, USA)
CPU	AMD Ryzen 9 7950X (Advanced Micro Devices, Inc., Santa Clara, CA, USA)
GPU	NVIDIA GeForce RTX 4090 (ASUSTeK Computer Inc., Taipei, Taiwan)
RAM	32GB (Samsung Electronics Co., Ltd., Suwon, Republic of South Korea)

3.5. Evaluation Metrics

Model performance was evaluated using metrics commonly adopted in RUL prediction, with RMSE and the asymmetric C-MAPSS score as the primary measures. MAE and R^2 were included as secondary reference metrics, and training efficiency was assessed using the average training time per epoch.

- RMSE (root mean squared error): Measures the average magnitude of the error between predicted and true RUL; lower values indicate better performance [30].

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y}_i)^2} \tag{1}$$

- C-MAPSS Score (asymmetric score): Penalizes late predictions more heavily than early ones, reflecting the higher risk associated with overestimating RUL. For each test sample, lower scores indicate better performance [31].

$$Score = \sum_{i=1}^{n} s_i,\ s_i = \begin{cases} e^{-\frac{(y_i - \widehat{y}_i)}{13}} - 1, \text{ if } (y_i - \widehat{y}_i) < 0 \\ e^{\frac{(y_i - \widehat{y}_i)}{10}} - 1, \text{ if } (y_i - \widehat{y}_i) \geq 0 \end{cases} \tag{2}$$

- MAE (mean absolute error): Measures the average absolute difference between predicted and true RUL; lower values indicate better performance [3].
- R^2 (coefficient of determination): Represents the proportion of variance in the true RUL that is explained by the predictions; values closer to 1 indicate a better fit [5].
- Average training time per epoch: Computed over the supervised AE-LSTM training stage and does not include the one-time 50-epoch AE pre-training overhead. Throughout the remainder of the paper, this quantity is used as the main indicator of training efficiency [9].

For each window-batch configuration, all metrics were computed on the test set for each random seed. The results were then aggregated across seeds by reporting the mean values and 95% confidence intervals obtained via bootstrap resampling [9]. In addition, average training time per epoch was recorded for each configuration to analyze the trade-offs between prediction accuracy and training efficiency. To assess whether performance differences between neighboring window sizes were statistically detectable at our sample size, Wilcoxon signed-rank tests were applied to paired RMSE values for the best and second-best windows at each batch size and encoder mode [9,11,32]. Because each configuration was evaluated with only five random seeds, these tests have limited statistical power and are therefore used as supportive evidence rather than as definitive proof of equivalence between window sizes.

4. Results

4.1. Results on FD001

Experiments on the FD001 subset were conducted over a grid of window sizes, batch sizes, and encoder training modes (frozen versus fine-tuned), with multiple random seeds for each configuration. For every setting, test RMSE, C-MAPSS score, MAE, R^2, and average training time per supervised epoch were computed, and mean values with 95% confidence intervals were obtained by bootstrap resampling [5,8,10].

On FD001, the best mean performance was obtained when the encoder was fine-tuned with a batch size of 128 and a window size of 64. In this configuration, the AE-LSTM model achieved an RMSE of 13.99 with a 95% confidence interval from 13.66 to 14.38, together with an R^2 of 0.887. The corresponding C-MAPSS score and MAE were also among the best across all settings, while the average supervised-stage training time per epoch was only 0.75 s. Detailed results for the best configuration of each batch size and encoder mode are summarized in Table 4, which provides a compact overview of the achievable accuracy–efficiency trade-offs on FD001 under the fixed architecture considered in this study.

Table 4. Summary of Experimental Results for FD001 Dataset.

Batch Size	Encoder Mode	Best Window	RMSE	C-MAPSS Score	MAE	R^2	Time (s) *
32	finetune	48	14.13	309.64	10.49	0.884	2.67
	frozen	70	14.38	353.96	11.04	0.880	2.09
64	finetune	26	14.29	323.44	10.46	0.882	1.55
	frozen	68	14.43	347.05	10.96	0.879	1.17
128	finetune	64	13.99	326.78	10.46	0.887	0.75
	frozen	68	14.34	329.42	10.88	0.881	0.68
256	finetune	90	14.45	350.26	10.74	0.879	0.51
	frozen	36	14.68	386.03	10.76	0.875	0.43
512	finetune	14	24.26	2280.5	18.63	0.609	0.28
	frozen	40	26.18	3614.55	21.8	0.511	0.31

* Average training time per epoch.

4.2. Effect of Window Size on FD001

The effect of window size was examined by plotting RMSE as a function of window length for each batch size and encoder mode. Window sizes from 10 to 100 cycles (step 2) were considered, and each point represents the average over several random seeds.

When the window is very short (20 cycles or less), all batch sizes yield noticeably higher errors, indicating that the temporal context is insufficient to capture early degradation signatures and longer-term trends. As the window length increases into a moderate range, roughly between 40 and 70 cycles, RMSE decreases and then remains relatively stable. For example, with a fine-tuned encoder, the best configurations at batch sizes 32 and 128 use windows of 48 and 64 cycles, respectively (Table 4). For batch size 64, the global minimum RMSE occurs at a shorter window of 26 cycles, but several windows in the 40–70 cycle range yield nearly identical RMSE values, forming a broad plateau around the optimum. In this region, neighboring window sizes typically produce very similar RMSE values, and Wilcoxon signed-rank tests between the best and second-best windows for batch sizes 32–256 yield p-values of at least 0.625. Given the small number of seeds, this suggests that, at our sample size, we do not observe strong evidence of a difference between adjacent windows at the 5% level [9].

For very long windows (80 cycles and above), performance becomes less consistent. Some configurations, such as batch 256 with a window of 90 cycles, still exhibit competitive accuracy, but in general, the error tends to increase slightly, and variability across seeds becomes larger. This behavior is consistent with the reduction in effective sample size and batch diversity when long sequences are used.

These results are shown in Figure 3, which shows the RMSE curve over window size for batch size 128 under the fine-tuned encoder setting; the shaded area highlights the near-plateau region around 40–70 cycles.

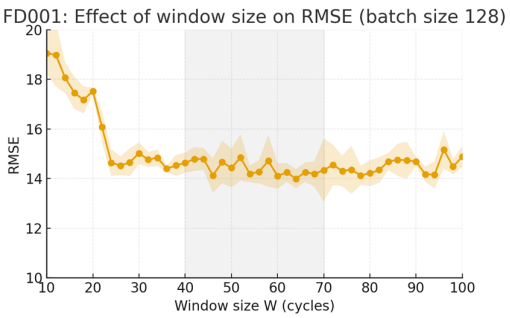


Figure 3. FD001: Effect of window size on RMSE (best-performing configuration on batch 128, fine-tuned encoder).

4.3. Effect of Batch Size and Training Efficiency on FD001

Batch size has a direct impact on both generalization performance and computational cost [12]. To analyze this trade-off, the best configuration for each batch size under the fine-tuned encoder setting was selected and compared.

Increasing the batch size from 32 to 128 leads to a clear gain in efficiency with only minor changes in accuracy. For instance, the best configuration at batch 32 (window 48) achieves an RMSE close to 14.1 but requires about 2.7 s per epoch, whereas the best configuration at batch 128 (window 64) reaches an RMSE of 13.99 with only 0.75 s per epoch. Batch 64 lies between these two extremes in both accuracy and speed. Further enlarging the batch size to 256 slightly increases RMSE (by less than half a point compared to batch 128) while reducing the average epoch time to about 0.5 s, which may be attractive when training time is a primary concern.

By contrast, at batch size 512, the best configuration on FD001 attains an RMSE of 24.26 and a C-MAPSS score above 2200, which is markedly worse than the values obtained for batch sizes 32–256 (RMSE around 14–15 and scores below 400; see Table 4). The corresponding confidence intervals are also much wider, indicating higher variability across seeds. Although this setting offers the shortest epoch time (< 0.3 s), it therefore lies far from the empirical accuracy–efficiency Pareto front [4]. This degradation suggests that the optimization dynamics were constrained by the learning rate schedule. As described in Section 3.4, an upper bound of 0.001 was imposed on the learning rate to ensure stability; for batch size 512, this cap prevented the learning rate from scaling linearly as intended, likely causing the optimizer to converge to sharp, suboptimal minima rather than generalizing well [12,13].

The relationship between RMSE and average epoch time for the best configuration at each batch size is depicted in Figure 4.

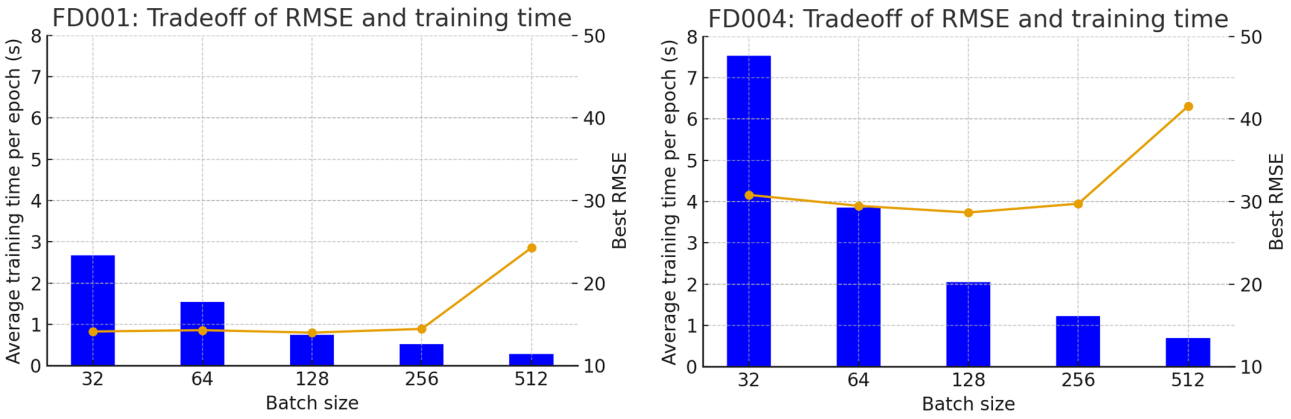


Figure 4. Trade-off between RMSE and training time across batch sizes using the fine-tuned encoder strategy. The blue bars represent the average training time per epoch, and the orange line indicates the best RMSE. (**a**) FD001 dataset, (**b**) FD004 dataset.

4.4. Frozen Versus Fine-Tuned Encoder on FD001

Two encoder training strategies were compared: keeping the AE encoder frozen after unsupervised pre-training and fine-tuning the encoder jointly with the LSTM head during supervised training [5,22]. For each batch size, the best configuration under each strategy was identified based on mean RMSE.

Across batch sizes from 32 to 256, the fine-tuned encoder consistently exhibited a lower mean RMSE than the frozen encoder (Table 4). The magnitude of the improvement is modest but robust across seeds and is accompanied by similar or slightly better C-MAPSS scores and MAE. The additional computational cost of fine-tuning the encoder is relatively small: for example, at batch size 128, the average epoch time increases by less than 0.1 s when encoder parameters are updated. Because only five seeds per configuration are available and no formal hypothesis tests are reported for these frozen-versus-fine-tuned comparisons, these observations should be viewed as descriptive trends rather than conclusive statistical evidence.

These observations suggest that, under the experimental conditions considered here, fine-tuning the encoder provides a favorable balance between accuracy and efficiency and can be recommended as a default choice for AE-LSTM-based RUL prediction [4,5,13].

4.5. Results on FD004

FD004 contains six operating conditions and two fault modes and therefore represents a more challenging RUL prediction scenario than FD001. To assess the robustness of the observations made on FD001, the same AE-LSTM architecture and training protocol were applied to FD004 without any additional dataset-specific tuning. All model hyperparameters, including the AE structure, optimizer, learning-rate schedule, and training epochs, were kept identical to those used for FD001, and the same grid of window sizes (10–100 cycles) and batch sizes (32, 64, 128, 256, 512) was employed.

On FD004, the best mean performance was again obtained with a fine-tuned encoder and a moderate batch size. The configuration with a window size of 64 and a batch size of 128 achieved a mean RMSE of 29.74 with a 95% confidence interval of 27.74–29.60, an MAE of 21.08, and an R^2 of 0.72, while requiring on average 2.05 s per training epoch. The corresponding C-MAPSS score was approximately 27,300. Compared with the best FD001 configuration (RMSE 13.99, $R^2 \approx 0.89$), the RMSE on FD004 is roughly doubled and R^2 is reduced, which is consistent with the increased difficulty of the multi-condition, multi-fault setting. Detailed results for the best configuration at each batch size on FD004 are summarized in Table 5.

Table 5. Summary of Experimental Results for FD004 Dataset.

Batch Size	Encoder Mode	Best Window	RMSE	C-MAPSS Score	MAE	R^2	Time (s) *
32	finetune	69	30.12	49,372.8	22.21	0.712	7.56
	frozen	42	40.27	79,524.1	31.24	0.501	7.41
64	finetune	74	29.48	45,446.0	21.92	0.707	3.85
	frozen	12	38.76	82,798.1	30.23	0.493	3.74
128	finetune	76	28.67	27,303.7	21.08	0.723	2.05
	frozen	10	38.75	65,700.6	30.17	0.494	1.95
256	finetune	68	29.74	32,741.4	22.13	0.702	1.23
	frozen	20	40.45	80,094.1	31.71	0.449	1.08
512	finetune	26	41.54	107,266.3	33.65	0.405	0.69
	frozen	36	46.25	388,160.5	37.55	0.280	0.64

* Average training time per epoch.

The dependence on window size for FD004 closely mirrors the trends observed on FD001, but with a tendency toward slightly longer optimal windows. For batch sizes 64, 128, and 256 under the fine-tuned encoder, the best-performing windows are found in the range between 68 and 76 cycles, with RMSE values clustered around 29–30. Very short windows (10–20 cycles) lead to clearly higher errors, indicating insufficient temporal context to capture degradation across multiple operating conditions [4,8]. As the window length increases into the moderate range (approximately 60–80 cycles), RMSE decreases and then remains relatively stable, and neighboring window sizes around the optimum show only small differences in performance. Very long windows beyond 80 cycles result in a slight deterioration of accuracy and increased variability across seeds, consistent with the reduction in effective sample size and batch diversity. These patterns are shown in Figure 5, where the RMSE-window curves for FD004 exhibit a broad plateau region similar to that of FD001.

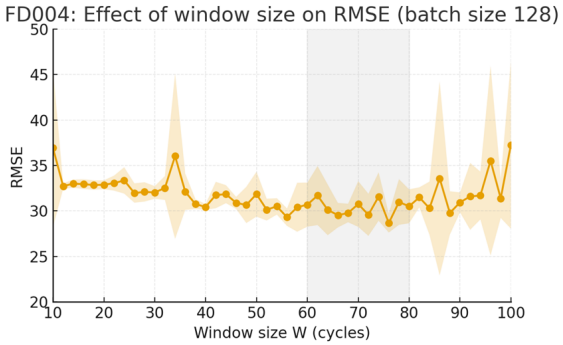


Figure 5. FD004: Effect of window size on RMSE (best-performing configuration on batch 128, fine-tuned encoder).

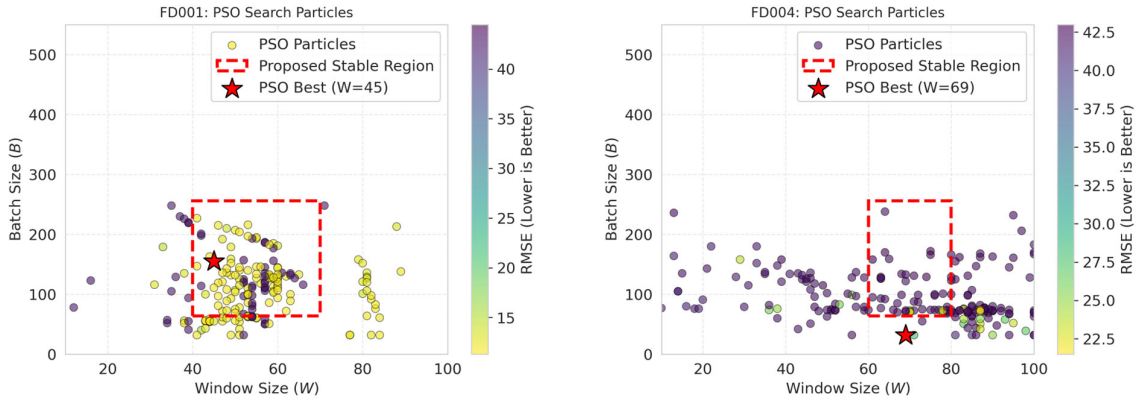
The effect of batch size on FD004 also follows the same accuracy–efficiency trade-off as on FD001. For batch sizes 64, 128, and 256 with a fine-tuned encoder, RMSE remains in a relatively narrow band around 29–30, while the average training time per epoch decreases from about 3.85 s at batch 64 to about 1.23 s at batch 256. In contrast, a batch size of 512 leads to a substantial degradation in generalization, with the best configuration yielding an RMSE of 41.54 and an R^2 of 0.40, despite achieving the fastest training time (approximately 0.69 s per epoch) and exhibiting large confidence intervals and high variance across seeds. This behavior is consistent with the FD001 results and with theoretical findings on the adverse effect of very large batches on generalization. The joint relationship between RMSE and training time for FD001 and FD004 is summarized in Figure 4.

Fine-tuning the encoder is particularly beneficial on FD004. For batch sizes 64, 128, and 256, the best frozen-encoder configurations exhibit RMSE values between approximately 38.7 and 40.5, whereas the corresponding fine-tuned configurations reduce RMSE to the 28–30 range, improving accuracy by about 9–11 points and increasing R^2 by roughly 0.20–0.25. The additional computational cost of fine-tuning the encoder remains moderate, as the increase in epoch time is small compared with the overall variation across batch sizes [4,6]. Table 5 summarizes the comparison between frozen and fine-tuned encoder strategies for FD004, highlighting that encoder fine-tuning is even more advantageous in the more complex FD004 scenario.

4.6. Validation via Intelligent Optimization

To validate the hyperparameter patterns observed in the grid search, we conducted an additional optimization experiment using PSO. While grid search provides a comprehensive landscape of the performance, it is computationally expensive and limited to discrete steps [15]. PSO, in contrast, explores the continuous search space to locate global optima efficiently [15]. We configured the PSO with 20 particles and 20 iterations for both FD001 and FD004 subsets to find the optimal window size and batch size. Figure 6 visualizes the search history of the PSO particles alongside the stable regions identified by our grid search.

For FD001 (Figure 6a), the global optimum found by PSO was a window size of 45 and batch size of 155, achieving a best-run RMSE of 11.11. This result falls precisely within the center of the stable region identified by our grid search (window sizes ranging from 40 to 70 and batch sizes from 64 to 256). The convergence of PSO particles into this region empirically confirms that our proposed range represents a robust global optimum for simple operating conditions.



(**a**) (**b**)

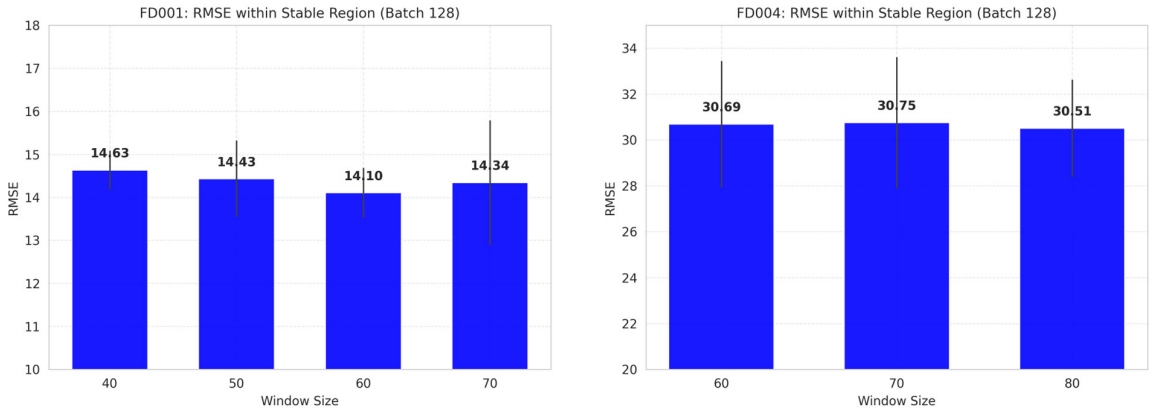
Figure 6. Validation of stable hyperparameter regions using PSO. (**a**) FD001 dataset: PSO best result is $W = 45$, $B = 155$, (**b**) FD004 dataset: PSO best result is $W = 69$, $B = 32$.

For FD004 (Figure 6b), PSO identified a window size of 69 and a batch size of 32. The optimal window size of 69 aligns perfectly with our recommended range (window sizes between 60 and 80), corroborating our finding that complex datasets with multiple fault modes require a sufficiently long temporal context to capture degradation trends effectively. Regarding the batch size, although the PSO-selected value of 32 is smaller than our recommended range (64–256), the performance difference is negligible in practice. Our grid search analysis revealed that increasing the batch size from 32 to 128 results in a comparable mean RMSE, while significantly reducing the training time per epoch. Therefore, while PSO efficiently locates a specific optimum, our grid search results provide the critical 'stability map' required for robust industrial deployment, confirming that the proposed range offers a practical balance of accuracy and efficiency. Overall, the validation confirms that the stable plateaus identified in this study are reliable design zones that encompass global optima [12].

4.7. Experimental Analysis

The experiments on FD001 and FD004 provide a coherent picture of how window size, batch size, and encoder training strategy jointly affect AE-LSTM-based RUL prediction. Although FD004 is substantially more difficult than FD001 due to its multiple operating conditions and fault modes, both subsets exhibit similar qualitative patterns.

For the window size, very short windows lead to clearly inferior performance on both datasets, indicating that a limited temporal context is insufficient to represent long-term degradation behavior [32]. As the window length increases, RMSE decreases and then forms a broad plateau of near-optimal performance. Figure 7 quantitatively illustrates this stability by comparing the mean RMSE of representative window sizes within these optimal ranges (fixed at Batch 128). On FD001, this plateau appears roughly between 40 and 70 cycles, while FD004 favors slightly longer windows of about 60 to 80 cycles. The overlapping error bars in Figure 7 indicate that the performance differences within these ranges are statistically negligible. Within these ranges, neighboring window sizes show only minor differences, and statistical tests, while limited by the small number of seeds, do not reveal significant gaps between the best and second-best windows. These observations suggest that, once a sufficient temporal context is provided, the AE-LSTM model is relatively robust to moderate changes in the window length. This insight allows for practical flexibility: practitioners can prioritize faster responsiveness by selecting a smaller window (e.g., 40) or enhance noise robustness with a larger window (e.g., 70) without compromising prediction accuracy [10,12].



(**a**) (**b**)

Figure 7. RMSE stability within the identified optimal window ranges (Batch size fixed at 128). Error bars (Black line) indicate standard deviation. (**a**) FD001 dataset, (**b**) FD004 dataset.

Batch size affects both generalization and training efficiency. On both FD001 and FD004, batch sizes between 64 and 256 provide the most favorable accuracy–efficiency trade-offs. In this regime, increasing the batch size reduces the average training time per epoch without substantially degrading RMSE or C-MAPSS score. In particular, a batch size of 128 yields the best overall accuracy on FD001 and competitive performance on FD004, while a batch size of 256 offers additional speed-up at the cost of a small accuracy loss. In contrast, very large batches (512) consistently produce the fastest training but also the worst generalization performance, with markedly higher errors and wider confidence intervals. This behavior is in line with theoretical findings that associate very large batches with sharp minima and poorer generalization [12].

The comparison between frozen and fine-tuned encoder strategies shows that adapting the encoder during supervised training is beneficial in all cases and especially important on FD004. While the improvement from fine-tuning is modest but consistent on FD001, the gains become substantial on FD004, where RMSE decreases by several points and R^2 increases significantly compared with the frozen-encoder baseline. This indicates that unsupervised pre-training alone is not sufficient in heterogeneous, multi-condition settings, and that supervised adaptation of the latent representations plays a key role [5].

It is also significant that FD004 was evaluated without any additional tuning beyond the settings chosen for FD001. Despite this constraint, the same overall hyperparameter patterns emerge: extremely short windows and very large batches should be avoided; moderate windows and batch sizes form broad regions of near-optimal performance; and encoder fine-tuning systematically improves results. This consistency highlights the complementary value of the two optimization approaches. While intelligent algorithms like PSO (Section 4.6) efficiently pinpoint specific global optima, the comprehensive grid search reveals the underlying performance landscape. It confirms that optimal solutions are not isolated peaks but broad, stable regions. This understanding empowers practitioners to make informed trade-offs—such as prioritizing training speed or inference latency—by selecting configurations from these robust zones rather than rigidly adhering to a single hyper-optimized value [10,12]. These consistent trends across two datasets of differing complexity support the conclusion that window size, batch size, and encoder training strategy are primary drivers of performance and efficiency in AE-LSTM-based RUL prediction. With appropriate choices of these hyperparameters, the relatively simple AE-LSTM architecture considered here achieves reasonably strong results on both simple and complex C-MAPSS subsets, although a noticeable performance gap to the best reported FD001 models in the literature remains.

To put the FD001 results into context, Table 6 summarizes RMSE values reported by representative recent models together with the AE-LSTM result from this study. All literature values in Table 6 were taken directly from the corresponding publications under the standard C-MAPSS evaluation protocol with training-label cap 125 cycles and were not re-implemented here.

Table 6. Comparison of C-MAPSS FD001 RUL models.

Reference	Model Structure	RMSE	Sliding Windows Size
Proposed	AE-LSTM	13.99	40–70
Shen et al. [3]	GWO-1DCNN	13.76	22
Fan et al. [32]	BiLSTM-DAE-Transformer	10.98	30
	LSTM	16.14	30
Tan and Teo. [33]	CNN-ATT	10.60	30

5. Discussion

5.1. Impact of Window Size and Physical Interpretation

The results show that window size is a critical temporal hyperparameter. Very short windows consistently underperform, indicating that a limited temporal context cannot represent long-term degradation patterns. Once the window length enters a moderate regime, a broad plateau of near-optimal performance emerges—approximately 40–70 cycles for FD001 and 60–80 cycles for FD004. Within these ranges, neighboring window sizes yield similar RMSE values, and formal tests based on five seeds do not detect statistically significant differences between the best and second-best windows, although this negative result should be interpreted cautiously because of the limited sample size [9].

These observations on window size can be interpreted through the physical degradation process of the equipment, consistent with recent studies on adaptive RUL prediction [4,34]. As identified in the literature, degradation typically evolves through stages such as run-in, linear aging, and nonlinear aging, each requiring different temporal contexts [5]. The poor performance at short window sizes ($W < 30$) aligns with the need to capture long-term dependencies; turbofan engine sensors often contain high-frequency vibration noise and operational fluctuations, so without sufficient historical context, the model fails to distinguish between sensor noise and incipient degradation trends. Conversely, the performance drop at very long windows ($W > 80$) can be attributed to the 'smoothing effect' during the nonlinear aging stage. As emphasized in Thakuri et al. [34], rapid degradation in the late stage requires the model to be sensitive to recent changes. During rapid failure propagation (e.g., crack growth), an excessively long fixed window introduces too much history from the stable linear phase, diluting the signal of the impending failure. The observed optimal plateau (40–70 cycles) thus represents a physical 'sweet spot' that balances these conflicting requirements [4].

5.2. Batch Size and Optimization Dynamics

Batch size manages both generalization and training efficiency. The results indicate that moderate batch sizes between 64 and 256 provide favorable accuracy–efficiency trade-offs on both subsets. In this range, increasing the batch size substantially reduces training time per epoch while maintaining similar RMSE and C-MAPSS scores, with batch sizes 128 and 256 forming an empirical Pareto front. In contrast, batch size 512 yields the fastest training but significantly worse generalization, with higher errors and wider confidence intervals. This pattern is consistent with prior findings that very large batches can drive optimization towards sharp minima with poor generalization [12].

Encoder training strategy further modulates these effects. Keeping the AE encoder frozen after unsupervised pre-training results in reasonable performance, but fine-tuning the encoder jointly with the LSTM head systematically improves accuracy, especially on FD004. The gains are modest but consistent on FD001 and become substantial on FD004, where multiple operating conditions and fault modes must be handled. This suggests that unsupervised pre-training alone is not sufficient in heterogeneous settings and that supervised adaptation of latent representations is particularly important when operating conditions vary [19,35].

5.3. Comparison with State-of-the-Art Models

Table 6 compares the proposed AE-LSTM with recent state-of-the-art models on the FD001 dataset, summarizing both their predictive performance (RMSE) and the strategies used to select the temporal context (window size). Compared with more complex architectures such as GWO-1DCNN and BiLSTM-DAE-Transformer models, the proposed AE-LSTM exhibits a moderate performance gap (e.g., RMSE 13.99 versus 10.98 or 13.76) [3,32]. Tan and Teo. [33] reported a superior RMSE of 10.60 on FD001 using a temporal CNN with attention mechanisms. While these advanced models achieve higher accuracy using specialized mechanisms, they often require more complex implementation compared to standard baselines.

However, a critical distinction lies in the hyperparameter selection strategy. As detailed in Table 6, many advanced models, such as Fan et al. [32] and Tan et al. [33], adopt a fixed window size of 30 cycles based on empirical conventions without explicitly analyzing the sensitivity of the model to this parameter. Even when meta-heuristic algorithms like GWO are employed—as in Shen et al. [3], where a specific window size of 22 was identified—the approach often functions as a "black box," yielding a single optimal point without revealing the surrounding performance landscape. In contrast, our systematic grid search clarifies that the optimal temporal context is not a single point but a broad "stable region" (e.g., 40–70 cycles for FD001).

This comparison clarifies the contribution of individual components: while advanced architectural mechanisms provide the final margin of accuracy, a significant portion of model performance is driven by the fundamental choice of temporal context (window size) and optimization dynamics (batch size). By tuning these components within the identified stable regions, even a simple AE-LSTM achieves competitive performance. Nevertheless, the combination of a simple architecture and a detailed hyperparameter study provides a transparent reference point and clarifies configuration regimes in which the AE-LSTM remains attractive when model complexity or implementation effort must be kept low. Furthermore, the performance gains observed in this study indicate the possibility that systematic hyperparameter optimization could offer additional improvements for more complex architectures as well.

5.4. Insights from Validation via Intelligent Optimization

These findings also underscore the complementary value of combining systematic grid search with intelligent optimization. Our PSO experiment served as a rigorous validation, as the identified global optima converged precisely within the stable regions derived from the grid search. While PSO is highly effective for efficiently pinpointing a specific optimal configuration [36], the grid search contextualizes this point within a broader "performance landscape." By leveraging both perspectives, we confirmed the existence of verified "stable plateaus"—such as the 40–70 cycle range for FD001. This offers a significant practical advantage: rather than rigidly adhering to a single hyper-optimized value found by an algorithm, designers can flexibly adjust the window size within this verified range to satisfy

specific system constraints—such as prioritizing rapid responsiveness or maximizing noise robustness—while ensuring near-optimal model performance [37,38].

5.5. Limitations and Future Directions

Several limitations should be acknowledged. First, the analysis is restricted to one model family—a fully connected AE with an LSTM head—and to two subsets of a single benchmark dataset. Other architectures, such as convolutional or attention-based models, and additional datasets from different industrial domains, may exhibit different sensitivity to window and batch size. Second, only window size and batch size were systematically varied, while other hyperparameters such as the form of the learning-rate schedule and network capacity were kept fixed. The interaction between batch size and learning rate is a well-known factor in optimization dynamics, and fixing the learning-rate schedule might have influenced the optimal batch sizes observed in this study [13,39]. For example, the strong degradation at batch size 512 may partly reflect this choice. Third, each configuration was evaluated with only five random seeds, so bootstrap confidence intervals and Wilcoxon tests should be interpreted as providing indicative rather than definitive statistical evidence [9]. Fourth, while Table 6 provides a brief comparison with representative FD001 baselines from the literature, no analogous external baseline comparison is reported for FD004 because of heterogeneous experimental protocols across prior works. Fifth, while PSO proved effective in efficiently locating high-performance configurations, it inherently focuses on pinpointing specific optimal points rather than explicitly identifying the broader "stable regions". Conversely, the exhaustive grid search employed here successfully visualized these robust landscapes but required significant computational resources due to its discrete and extensive nature. To address this trade-off, future research should explore advanced optimization strategies—such as uncertainty-aware Bayesian optimization or region-based meta-heuristics—that can efficiently delineate optimal stability boundaries without the high computational cost of a full grid search. Finally, we did not include a direct ablation study comparing the AE-LSTM pipeline with purely supervised alternatives (e.g., an LSTM trained directly on the 17-dimensional inputs or an AE encoder initialized randomly and trained end-to-end). As a result, the incremental benefit of unsupervised pre-training relative to simpler baselines is not quantified here and remains an important topic for future work.

6. Conclusions

This study examined how window size, batch size, and encoder training strategy influence the performance and efficiency—measured in terms of the average supervised-stage training time per epoch—of an AE-LSTM RUL prediction model on the C-MAPSS FD001 and FD004 subsets. Using fixed and relatively simple architecture, we conducted multi-seed grid experiments over a wide range of window and batch sizes and applied the same protocol to both subsets without additional tuning for FD004.

Across both datasets, three consistent patterns emerged. First, very short windows underperform, whereas moderate window lengths yield a broad plateau of near-optimal performance. Second, batch sizes between 64 and 256 provide favorable accuracy–efficiency trade-offs, while extremely large batches (512) offer faster training at the cost of much worse generalization. Third, fine-tuning the encoder systematically improves accuracy, particularly in the more heterogeneous FD004 scenario.

Based on the RMSE vs. Window Size analyses presented in Figures 3 and 5, we propose the following practical configuration guidelines for AE-LSTM-based RUL prediction models:

FD001-like scenarios (single operating condition):

- Window size: approximately 40–70 cycles.

- Batch size: between 64 and 256.
- Encoder training: fine-tune the encoder together with the LSTM head.

FD004-like scenarios (multi-condition, multi-fault):

- Window size: approximately 60–80 cycles.
- Batch size: again between 64 and 256, with batch size 128–256 forming an empirical accuracy–efficiency Pareto front.
- Encoder training: Encoder fine-tuning is especially important to adapt the latent space to heterogeneous operating conditions.

Author Contributions: Conceptualization, E.J. and Y.K.; methodology, E.J. and Y.K.; software, E.J.; validation, E.J. and D.J.; formal analysis, E.J., D.J. and Y.K.; investigation, E.J., D.J. and Y.K.; re-sources, D.J. and Y.K.; data curation, E.J. and D.J.; writing—original draft preparation, E.J. and Y.K.; writing—review and editing, E.J., D.J. and Y.K.; visualization, E.J.; project administration, Y.K.; supervision, Y.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Korean Government (MOTIE) through the Korea Institute for Advancement of Technology (KIAT) grant (RS-2025-02263458, HRD Program for Industrial Innovation).

Data Availability Statement: The data used in this study are publicly available in the NASA Prognostics Center of Excellence data repository. The C-MAPSS dataset can be found at: https://www.nasa.gov/content/prognostics-center-of-excellence-data-set-repository (accessed on 27 December 2025).

Conflicts of Interest: Author Donghwan Jin was employed by the company MyMeta Co., Ltd. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AE	Autoencoder
LSTM	Long short-term memory
AE-LSTM	Autoencoder–long short-term memory
CNN	Convolutional neural network
RUL	Remaining useful life
PdM	Predictive maintenance
PHM	Prognostics and health management
C-MAPSS	Commercial Modular Aero-Propulsion System Simulation
RMSE	Root mean squared error
MAE	Mean absolute error
HPC	High-pressure compressor
IoT	Internet of Things
PSO	Particle Swarm Optimization
SA	Simulated Annealing (SA)

References

1. Fischer, D.; Moder, P.; Ehm, H. Investigation of Predictive Maintenance for Semiconductor Manufacturing and Its Impacts on the Supply Chain. In Proceedings of the 2021 22nd IEEE International Conference on Industrial Technology (ICIT), Valencia, Spain, 10–12 March 2021; Volume 1, pp. 1409–1416. [CrossRef]
2. Nunes, P.; Santos, J.; Rocha, E. Challenges in Predictive Maintenance—A Review. *CIRP J. Manuf. Sci. Technol.* **2023**, *40*, 53–67. [CrossRef]
3. Shen, L.; Wang, Y.; Du, B.; Yang, H.; Fan, H. Remaining Useful Life Prediction of Aero-Engine Based on Improved GWO and 1DCNN. *Machines* **2025**, *13*, 583. [CrossRef]

4. Jiang, L.; Zhang, X.; Cao, H.; Zhang, Y. A Transformer-Based Framework with Historical Data Fusion for RUL Prediction. *Meas. Sci. Technol.* **2025**, *36*, 106103. [CrossRef]
5. Lodygowski, T.; Szrama, S. Unsupervised Classification and Remaining Useful Life Prediction for Turbofan Engines Using Autoencoders and Gaussian Mixture Models: A Comprehensive Framework for Predictive Maintenance. *Appl. Sci.* **2025**, *15*, 7884. [CrossRef]
6. Belay, M.A.; Blakseth, S.S.; Rasheed, A.; Salvo Rossi, P. Unsupervised Anomaly Detection for IoT-Based Multivariate Time Series: Existing Solutions, Performance Analysis and Future Directions. *Sensors* **2023**, *23*, 2844. [CrossRef]
7. Li, Z.; He, Q.; Li, J. A Survey of Deep Learning-Driven Architecture for Predictive Maintenance. *Eng. Appl. Artif. Intell.* **2024**, *133*, 108285. [CrossRef]
8. Elsherif, S.M.; Hafiz, B.; Makhlouf, M.A.; Farouk, O. A Deep Learning-Based Prognostic Approach for Predicting Turbofan Engine Degradation and Remaining Useful Life. *Sci. Rep.* **2025**, *15*, 26251. [CrossRef]
9. Bouthillier, X.; Delaunay, P.; Bronzi, M.; Trofimov, A.; Nichyporuk, B.; Szeto, J.; Vincent, P. Accounting for Variance in Machine Learning Benchmarks. *Proc. Mach. Learn. Syst.* **2021**, *3*, 747–769.
10. Wang, C.H.; Liu, J.Y. Integrating Feature Engineering with Deep Learning to Conduct Diagnostic and Predictive Analytics for Turbofan Engines. *Math. Probl. Eng.* **2022**, *2022*, 9930176. [CrossRef]
11. Wang, Z.; Dahouda, M.K.; Hwang, H.; Joe, I. Explanatory LSTM-AE-Based Anomaly Detection for Time Series Data in Marine Transportation. *IEEE Access* **2025**, *13*, 117308–117320. [CrossRef]
12. Keskar, N.S.; Mudigere, D.; Nocedal, J.; Smelyanskiy, M.; Tang, P.T.P. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. *arXiv* **2016**, arXiv:1609.04836. [CrossRef]
13. Fristiana, A.H.; Alfarozi, S.A.I.; Permanasari, A.E.; Pratama, M.; Wibirama, S. A Survey on Hyperparameters Optimization of Deep Learning for Time Series Classification. *IEEE Access* **2024**, *12*, 191162–191198. [CrossRef]
14. Almeida, J.; Soares, J.; Lezama, F.; Limmer, S.; Rodemann, T.; Vale, Z. A systematic review of explainability in computational intelligence for optimization. *Comput. Sci. Rev.* **2025**, *57*, 100764. [CrossRef]
15. Rajwar, K.; Deep, K.; Das, S. An Exhaustive Review of the Metaheuristic Algorithms for Search and Optimization: Taxonomy, applications, and open challenges. *Artif. Intell. Rev.* **2023**, *56*, 13187–13257. [CrossRef]
16. Li, G.; Jung, J.J. Deep Learning for Anomaly Detection in Multivariate Time Series: Approaches, Applications, and Challenges. *Inf. Fusion* **2023**, *91*, 93–102. [CrossRef]
17. Frederick, D.K. User's Guide for the Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) Software; NASA Technical Memorandum NASA/TM—2007-215026, 2007. Available online: https://ntrs.nasa.gov/api/citations/20070034949/downloads/20070034949.pdf (accessed on 13 October 2025).
18. DeCastro, J.A.; Litt, J.S.; Frederick, D.K. A Modular Aero-Propulsion System Simulation of a Large Commercial Aircraft Engine; NASA Technical Memorandum NASA/TM—2008-215303, 2008. Available online: https://ntrs.nasa.gov/api/citations/20080043619/downloads/20080043619.pdf (accessed on 13 October 2025).
19. Vollert, S.; Theissler, A. Challenges of Machine Learning-Based RUL Prognosis: A Review on NASA's C-MAPSS Data Set. In Proceedings of the 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Vasteras, Sweden, 7–10 September 2021; pp. 1–8. [CrossRef]
20. Mitici, M.; de Pater, I.; Barros, A.; Zeng, Z. Dynamic Predictive Maintenance for Multiple Components Using Data-Driven Probabilistic RUL Prognostics: The Case of Turbofan Engines. *Reliab. Eng. Syst. Saf.* **2023**, *234*, 109199. [CrossRef]
21. Chazhoor, A.; Mounika, Y.; Sarobin, M.V.R.; Sanjana, M.V.; Yasashvini, R. Predictive Maintenance Using Machine Learning-Based Classification Models. *IOP Conf. Ser. Mater. Sci. Eng.* **2020**, *954*, 012001. [CrossRef]
22. Hong, C.W.; Lee, C.; Lee, K.; Ko, M.S.; Kim, D.E.; Hur, K. Remaining Useful Life Prognosis for Turbofan Engine Using Explainable Deep Neural Networks with Dimensionality Reduction. *Sensors* **2020**, *20*, 6626. [CrossRef]
23. Kulanuwat, L.; Chantrapornchai, C.; Maleewong, M.; Wongchaisuwat, P.; Wimala, S.; Sarinnapakorn, K.; Boonya-Aroonnet, S. Anomaly Detection Using a Sliding Window Technique and Data Imputation with Machine Learning for Hydrological Time Series. *Water* **2021**, *13*, 1862. [CrossRef]
24. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.
25. Zamanzadeh Darban, Z.; Webb, G.I.; Pan, S.; Aggarwal, C.; Salehi, M. Deep Learning for Time Series Anomaly Detection: A Survey. *ACM Comput. Surv.* **2024**, *57*, 15. [CrossRef]
26. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention Is All You Need. *arXiv* **2017**. [CrossRef]
27. Ong, K.S.H.; Wang, W.; Niyato, D.; Friedrichs, T. Deep-Reinforcement-Learning-Based Predictive Maintenance Model for Effective Resource Management in Industrial IoT. *IEEE Internet Things J.* **2021**, *9*, 5173–5188. [CrossRef]

28. Zhang, C.; Song, D.; Chen, Y.; Feng, X.; Lumezanu, C.; Cheng, W.; Chawla, N.V. A Deep Neural Network for Unsupervised Anomaly Detection and Diagnosis in Multivariate Time Series Data. In Proceedings of the AAAI Conference on Artificial Intelligence, Honolulu, HI, USA, 27 January–1 February 2019; Volume 33, pp. 1409–1416. [CrossRef]
29. Yıldırım, U.; Afşer, H. Linear Methods for Predictive Maintenance: The Case of NASA C-MAPSS Datasets. *Appl. Sci.* **2025**, *15*, 9945. [CrossRef]
30. Hodson, T.O. Root-mean-square error (RMSE) or mean absolute error (MAE): When to use them or not. *Geosci. Model Dev.* **2022**, *15*, 5481–5487. [CrossRef]
31. Ramasso, E.; Saxena, A. Review and Analysis of Algorithmic Approaches Developed for Prognostics on CMAPSS Dataset. In Proceedings of the Annual Conference of the Prognostics and Health Management Society 2014, Fort Worth, TX, USA, 29 September–2 October 2014; Volume 6. [CrossRef]
32. Fan, Z.; Li, W.; Chang, K.-C. A Bidirectional Long Short-Term Memory Autoencoder Transformer for Remaining Useful Life Estimation. *Mathematics* **2023**, *11*, 4972. [CrossRef]
33. Tan, W.M.; Teo, T.H. Remaining Useful Life Prediction Using Temporal Convolution with Attention. *AI* **2021**, *2*, 48–70. [CrossRef]
34. Thakuri, S.K.; Li, H.; Ruan, D.; Wu, X. The RUL Prediction of Li-Ion Batteries Based on Adaptive LSTM. *J. Dyn. Monit. Diagn.* **2025**, *4*, 53–64. [CrossRef]
35. Leukel, J.; González, J.; Riekert, M. Machine Learning-Based Failure Prediction in Industrial Maintenance: Improving Performance by Sliding Window Selection. *Int. J. Qual. Reliab. Manag.* **2023**, *40*, 1449–1462. [CrossRef]
36. Zito, F.; Talbi, E.-G.; Cavallaro, C.; Cutello, V.; Pavone, M. Metaheuristics in automated machine learning: Strategies for optimization. *Intell. Syst. Appl.* **2025**, *26*, 200532. [CrossRef]
37. Bischl, B.; Binder, M.; Lang, M.; Pielok, T.; Richter, J.; Coors, S.; Lindauer, M. Hyperparameter optimization: Foundations, algorithms, best practices, and open challenges. *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* **2023**, *13*, e1484. [CrossRef]
38. Rakesh, V.; Mazumdar, S.; Samanta, T.; Pal, S.; Das, A. Impact of hyperparameter optimization on the accuracy of lightweight deep learning models for real-time image classification. *arXiv* **2025**, arXiv:2507.23315. [CrossRef]
39. Gupta, M.M. Fuzzy Logic and Neural Networks. In Proceedings of the 1992 IEEE International Conference on Systems Engineering, Kobe, Japan, 17–19 September 1992; pp. 636–639. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.